2019 Virtualized CPE Services Have Finally Arrived Via Service Delivery Platforms

Home Gateway Feature and Service Flexibility, Avoiding The Monolithic Upgrade Path

A Technical Paper prepared for SCTE•ISBE by

Ian Wheelock
Engineering Fellow, CPE Solutions
CommScope
4300 Cork Airport Business Park, Kinsale Road, Cork, Ireland
00353-86-235-2712
Ian.Wheelock@CommScope.com

Charles Cheevers,
CTO CPE Solutions
CommScope
3871 Lakefield Dr, Suwanee, GA 30024
678-473-8507
Charles.Cheevers@CommScope.com

Table of Contents

List of Figures

Introduction

Current models for adding new services and features to the home are highly reliant on upgrading gateway devices with a monolithic firmware image. Typically, lots of effort is required from the Cable Operator, the gateway Original Equipment Manufacturer (OEM), and possibly a 3rd party Software supplier to add these new features. This not only involves the specification of how everything should fit together, but the planning, development, and testing of the new feature, as well as the entire monolithic firmware deliverable. As one can imagine the time and effort involved can be considerable. Once the monolithic image is created and deployed, the whole cycle restarts with the next feature or service the Cable Operator would like to deploy.

This model has worked. However, when compared to mobile phones or laptops, adding new software features typically does not require an OS upgrade. Why can't gateways follow this model? or use something a lot more agile that has fewer moving parts to enable faster feature and service delivery to subscribers?

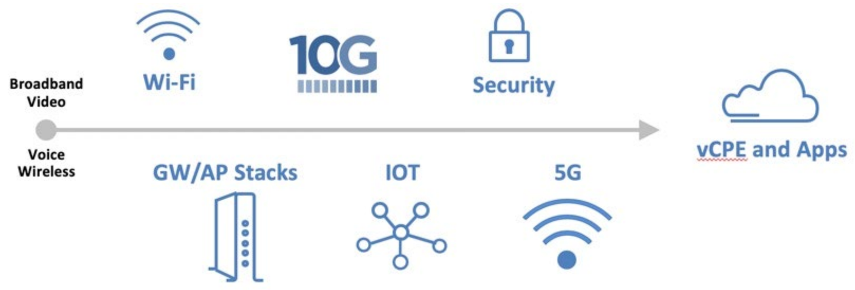

Figure 1 - Factors Driving New Software Services

This paper will concentrate on exploring what architectures and platform options exist today to deal with service delivery beyond the monolithic image system and examine the pros and cons of these including how virtualization techniques both in the gateway and in the cloud can be used. Details relating to RAM, flash, and CPU resources will also be covered. The paper will also address aspects of cloud-based applications based on application traffic tunneling and compare these as potential alternatives to thicker gateway hosted services. Its organized as follows and covers the following sections:

- New software delivery options that are beginning to appear in the industry
- What they mean for operators, OEMs and 3rd party software/service vendors
- How they might be applied to existing and future gateway platforms
- Impact on RAM/flash/CPU resources
- How to manage or orchestrate these services
- The tradeoff between thick gateway services vs virtualized cloud services
- How gateway traffic filtering and tunneling enables these services

Content

1. Gateway Platforms

Gateway platforms today have not changed much compared to initial gateways in relation to how software is developed and deployed, however the demand for new software and services has increased, particularly the integration of 3rd party software and services into existing gateways.

1.1. Gateway Stack

The standard approach for a gateway platform is to target a set of routing/networking features typically tied to a release of Linux, and maybe replace some of these with additional or upgraded components depending on the operator requirements for a new release, and then mix in a bunch of management controls and logging options to be able to manage and troubleshoot the platform in the field. The various changes are developed and unit tested, then are all mixed together to produce a monolithic image. This image is then submitted to the OEM test teams for system testing before the final release candidate is made available to the cable operator. The cable operator then takes this new release and applies their own acceptance testing to this image before finally releasing to the field. Finally, the operator can begin to offer the new feature set to their subscriber base. Proprietary, RDK-B, and openWrt gateway platforms all follow this general model.

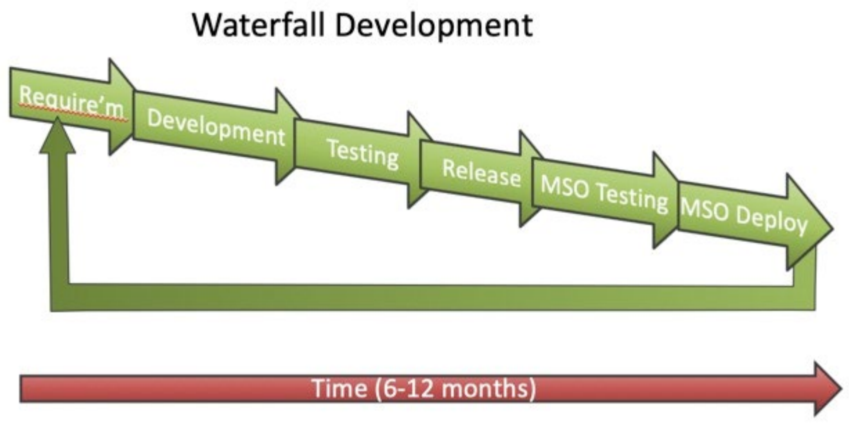

Figure 2 - Traditional Software Development Model

This big bang approach is repeated over and over as new features are requested, or improvements/bug fixes need to be incorporated. In some instances, depending on the type of change involved, more focused testing can be performed, getting the final monolithic upgrade ready for deployment. This lengthy development process include approach is typical within the industry, and has remained in place as a compromise to managing the risk of launching a completely new release out to thousands or millions of deployed gateways.

1.2. RDK

In most cases this development process is as streamlined as its going to get. Some new platforms like RDK enable more control around the build environment, and focus on continuous development and integration of new features, working towards constant integration and deployment of these into the field. This model depends on a lot of automated testing, significant logging support, and the ability to move from development directly into customer deployment on a regular cadence, perhaps once a month or even once a week. Given the exposure of regular updates of software in the field and knowing exactly what has changed from minor release to minor release, this model enables more focused testing and resolution of issues on specific features compared to the big bang omnibus release of features previously described.

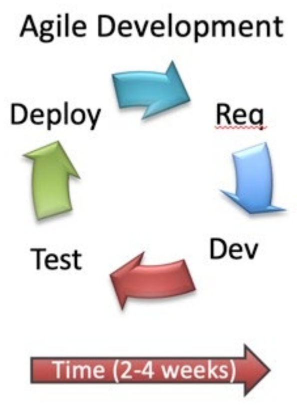

Figure 3 - Agile Development Model

In most cases where RDK has been deployed with a high release cadence, the operator involved has been tightly coupled with the actual development process, sharing bug tracking and build systems with OEMs they have partnered with, and requiring their own development teams to be able to guide the overall release planning/development of features as well as deal with issues arising from the field (performing triage, collection of logs, etc.). There are definite benefits from this high cadence approach in terms of quick turnaround of new features and bug fixes, but the model does require the cable operator to get down and dirty with the development process, as well as owning the release, system test and deployment processes.

All of this costs money by moving the operator into more of an OEM/development role. In most operator cases, developing software themselves is not how they make money from their business. The RDK codebase/architecture can still be used in the traditional development model, where an operator works with an OEM to release a set of new features and updates at a much lower cadence - maybe once every 6/9/12 months - while building on the stability of known RDK releases from the RDK community.

Both these approaches require either the OEM to do significant development or have the operator get tied into the development process, possibly at an uncomfortable level.

1.3. Adding 3rd Party Software

In the case of adding 3rd party software, generally there is a need to involve the software provider into the development process, one way or another. Such software maybe supplied in source code format or in binary/library format. For the binary/library format, the 3rd party typically requires access to the various code compiler elements to be able to cross compile their source code into a library that can be linked in to the monolithic firmware image.

This approach normally requires the OEM to develop target platform layer interfaces that the 3rd party library requires. If source code can be provided, this gets built directly by the OEM themselves, with the OEM still needing to develop the target platform layer, as well as any other management control/logging functions to fit into the existing platform. (The main reason source code is not normally shared is in order to protect any associated Intellectual Property Rights (IPR) from being exposed to OEMs/other parties)

Once the 3rd party software is integrated into a monolithic firmware image, it is subject to testing which typically involves the 3rd party vendor, the OEM and finally the operator acceptance testing. Again, a somewhat complicated setup.

1.4. Adding Operator User Interfaces

In the case of adding subscriber user interfaces, be they webpage based or mobile app based, the typical approach is for the operator to provide use-cases and some sample screen shots as part of a set of requirements to the OEM to implement. The user interface will typically have extra options included over time depending on what new software features maybe added. Such requirements are typically treated no differently than feature requirements, so being added to this lengthy process that results in a monolithic firmware image being produced. Unfortunately, user interfaces are very subjective due to the interpretation of the look and feel characteristics versus the actual implementation. Another issue with user interfaces is that an operator may have two OEM suppliers of gateways, or indeed have multiple language or countries where the number of OEMs increases but still needs an identical look and feel to apply to all gateways. Given the number of OEMs, expecting to get these independent software developers produce identical look and feel is a challenge.

1.5. Long Term Maintentnace

Overall, adding software features and services to existing gateways is typically complicated and quite involved, and in most cases has to be repeated for every different gateway an operator uses. Once such features and services are deployed, the time to fix issues or update to newer releases of the feature/service is determined more by the overall development and testing cycles rather than the availability of the fix/release, particularly with 6-12 month release cadences in use by a lot of operators. Such an approach can be extremely frustrating and significantly hampers feature velocity. If an operator choses to get involved in the development cycle of RDK-B, then it's possible to accelerate this, but one must remember that the gateway also has to preserve the robustness of all previous software features as well as any new features being developed/added, which is another cost to the development process.

Architecture for consideration

MSOs have a number of different devices that they use for broadband access, ranging from the most recent advanced gateways to older legacy products. Traditionally these devices sole purpose was to provide the networking platform for delivering broadband access to subscribers. In order to stay relevant and offer more than just a dumb internet pipe, MSO's have been evaluating how to enable new services for subscribers in a cost-effective way that can be run on existing devices in the field or augmented via hybrid cloud options. Given the mix of devices in the field, the target architectures need to cover services delivered solely on existing router devices or distributed in multiple places. A big challenge in achieving this is the range of devices in use, all with different capabilities as well as the type of expected service to offer.

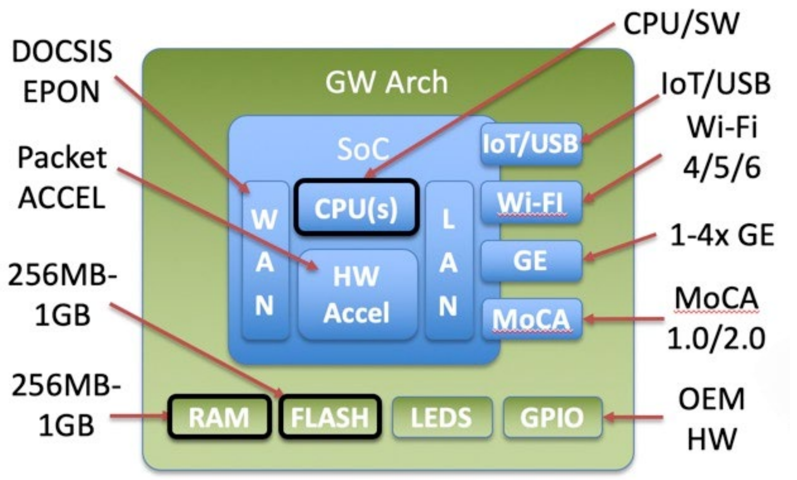

Figure 4 - Key HW Elements of Gateway

The following system architecture is proposed for MSO's to consider. This uses a mixture of container based orchestrated software services on in-home devices, as well as offering a hybrid option for services provided partially in the home and mostly in the cloud (through tunneling and the use of iptables or ovs on the in-home platform). The architecture also shows some tighter integration of services within the platform itself, for when some software needs higher performance access to networking or other lower layer services. The architecture can apply to most any WAN access, with D3.0, D3.1 and PON all shown.

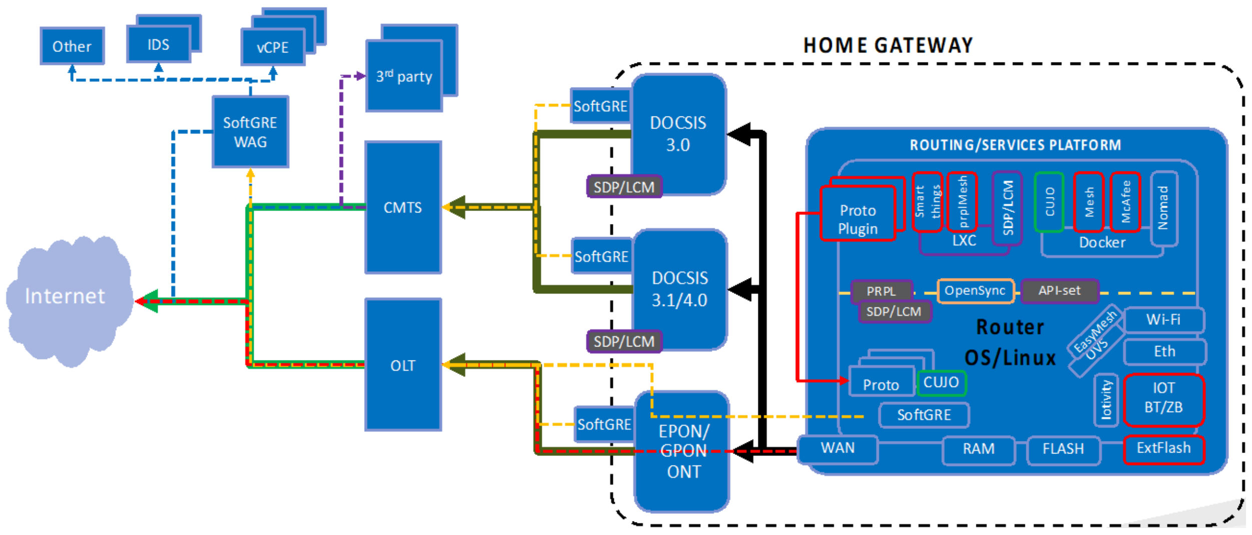

Figure 5 - Proposed home gateway, applicable to multiple access types

2. Variables

A lot of variables shape this architecture including: the device type; available RAM; available storage/flash (and read/write ability of same); the type of containers to be used; the number of services that may need to reside on the platform; whether these services are provided by the MSO directly, contracted partners, or third-party developers.

The types of new services are unlimited, with a huge array of options available to be added to the gateway platform

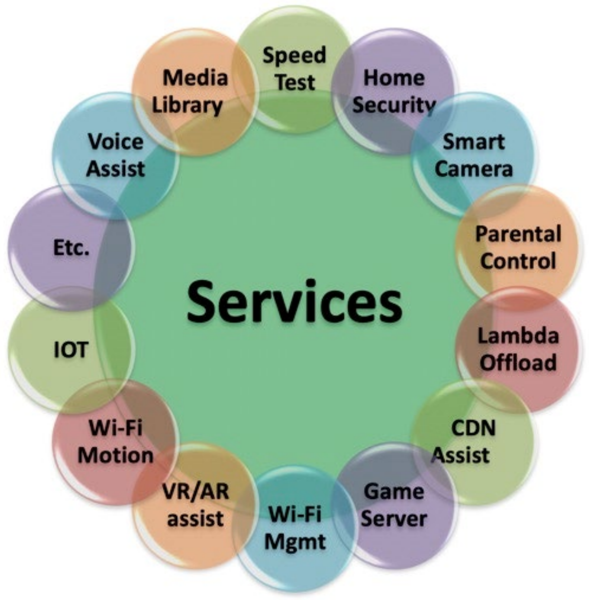

Figure 6 - Sample of Potential Services Considered

Other factors include: the supported services for the devices themselves; the types of low-level or high-level APIs available; the infrastructure for hosting services on the device; the types of services being considered (tightly integrated networking applications, or apps that only need IP connectivity); as well as what type of access to hardware or local software stack is needed and what the managed API interfaces to use are, etc.

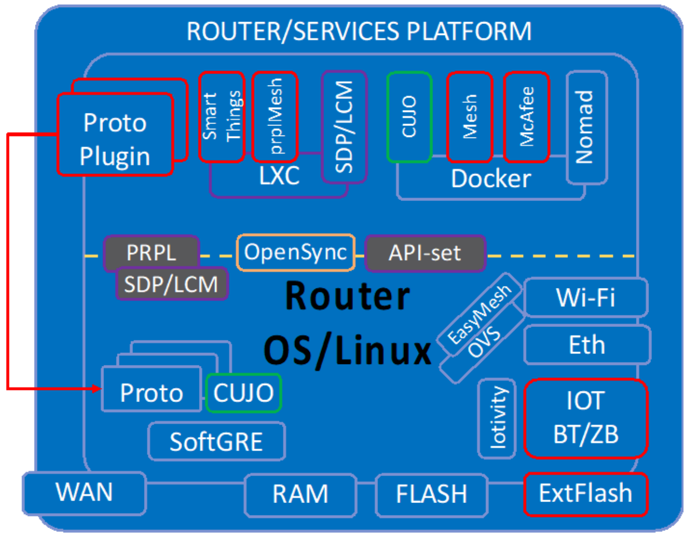

Figure 7 - Proposed Router/Services Platform close-up

Once the overall view of a containerized approach for local applications within devices is agreed, other decisions must be made regarding the types of container infrastructure to use, the type of orchestration involved, the ability to manage and monitor not only the individual device infrastructure but also the entire network of devices running containers, the overall performance of the system including initial deployment, upgrade and mass reconfiguration across the footprint of devices. There is also a need to consider the mixed management of normal day to day operation and maintenance of the deployed broadband system and these integrated services on the same infrastructure.

3. Device type, RAM and flash considerations

The deployed standalone devices have a variety of RAM and flash capabilities. Many platforms are quite limited, only including 512MB of RAM and 128MB of flash, while newer platforms are considering 512MB RAM and 512MB flash, or even 1GB RAM and 1GB flash. Cost is the main driver regarding how much storage to add to a device. A lot of purchasing decisions are made on the basis of the “hear and now”, as opposed to the total cost of ownership of the device and what feature upgradeability might be lost if too little memory is specified. For many years both RAM and flash costs were a large part of a gateway design, and something that could be manipulated with in the design, i.e. I need Wi-Fi, but maybe I could get away with 128MB instead of 256MB of RAM, particularly given price per MB. As a result, if an operator needed price reduction on a new design, RAM and flash were up for the chop. This made sense given 100,000s of device deployed, saving maybe $2-4 per unit is a big CAPEX save.

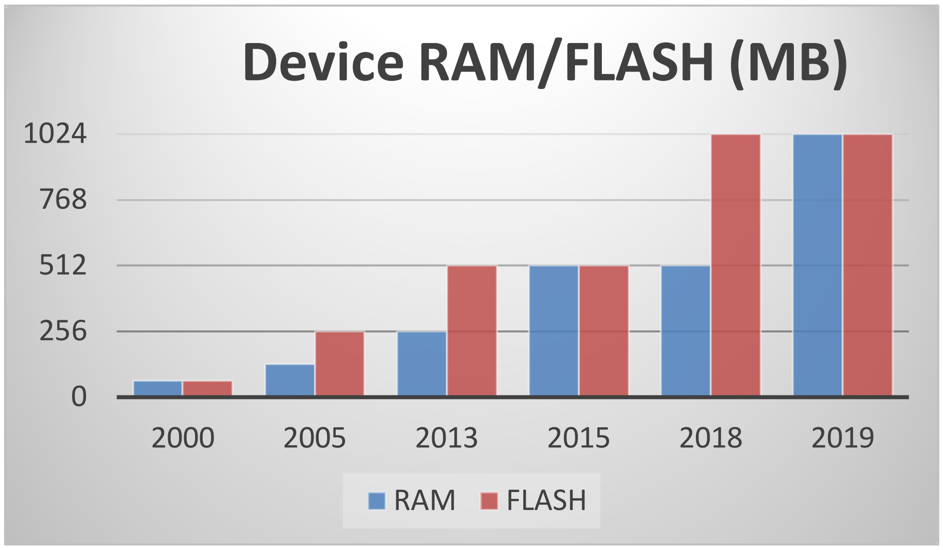

Figure 8 - Example GW Memory Trends

Depending on the routing software stack itself, most of this storage may already be consumed. Some firmware stacks using OpenWrt can actually be made to operate within 4MB flash and 32MB RAM, but in reality, need more like 128MB flash and 256MB RAM to fully support operator features.

3.1. Software deployement model

The typical software deployment model for broadband devices is to generate compressed squashFS firmware images (containing the full routing stack and any other software functions) that are distributed and stored in device flash, and during bootup of the device are decompressed from flash into RAM. In most cases the decompressed image itself is a soft copy of the platform Linux root filesystem (rootfs) which is presented in RAM to the OS. The OS will then launch applications from the rootfs.

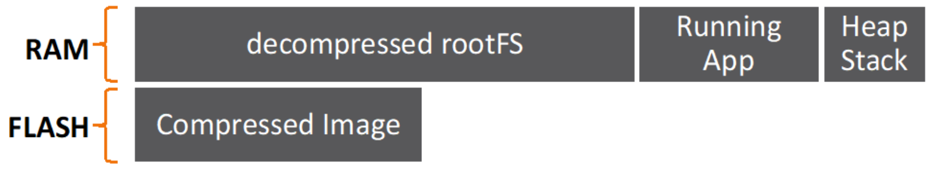

Figure 9 - Flash and RAM Organisation

Another decision made with broadband devices is to use a “dual image” option for firmware image storage, where two complete compressed images are stored in the flash storage, effectively limiting the maximum image size to under 50% of the available flash memory. This is done to have a backup image, in case an image has been corrupted in flash (due to various possible reasons).

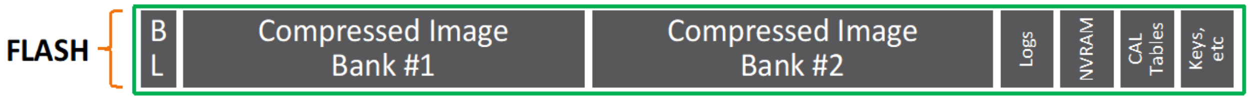

Figure 10 - Flash memory Organisation

Thankfully RAM and flash pricing has corrected over the last 2 years, meaning prices have come down (different reasons for RAM and flash). However, operator purchasing decisions regarding RAM and flash have had consequences on what feature upgrades may apply to existing deployed device, and in some cases, there just is no space left to factor in any local extensions or alternatives, and alternatives, such as hybrid or virtual cloud services must be considered.

Another aspect of broadband devices is that due to having a single monolithic firmware image containing all the software for the system to operate with, any minor changes requiring a complete replacement of this monolithic image. Even though this appears quite inefficient, there are a lot of operational benefits in knowing that a population of devices are running version #N or version #N-1 of firmware.

Most broadband devices limit flash storage to be READONLY, with only the bootloader or firmware upgrade process being able to write anything to flash. This is a major issue when considering the download and storage of software components separate to the main firmware image. In some platforms, read/write of flash is already supported, but other platforms may need bootloader/code refactoring to accommodate this mode of operation.

New software services packaged in containers (and similar) tend to be overlaid on top of the existing firmware image in some instances taking advantage of features/libraries within the platform image. Other software however may need to be integrated directly with the existing platform image, possibly replacing or adding functionality.

This idea of live patching of the platform itself brings considerable complexity and risk from the point of view of both modifying the actual system properly and ensuring that a patch does not cause any issue to the running system. Also, the management of a mixed population of devices that may have different levels of “patching” applied may present significant operational overhead.

3.2. Compression

Given the nature of compression, the firmware image is likely to be much larger in RAM when decompressed. The compressed firmware image is typically CRC/MD5/signature checked before any attempts to decompress/execute code to make sure the image has not suffered any corruption while resident in flash (or due to misprogramming) and that it is a proper cryptographically signed image. The firmware image includes the Linux kernel, drivers, and complete root filesystem. This system of compressing firmware images is the general approach used on all embedded platforms that use NAND flash memory, as it is not possible to execute directly from NAND, compared to NOR flash.

3.3. Compression, Flash and Containers

However, in a platform that may need to offer "container" based services, it may be better to consider separating how container images are stored and accessed in flash compared to the platform firmware image. Like most images, container images are compressed and, once downloaded, are accessed via squashFS. Isolating the container images to a separate flash partition/location will allow the container execution environment access the images directly from flash rather than requiring the complete container root filesystem to be copied to RAM. RAM is still required to load and run the various program files that comprise the container.

In nearly all these cases there is a need to support OverlayFS (a key Linux feature, in mainline since 3.18) to ensure any configuration elements or read-write locations are handled separately to the container read-only space in flash. This approach can reduce the overall amount of expensive RAM required (for storage purposes) on a platform at the cost of adding additional flash, and allows for flexibility in adding extra flash using either onboard eMMC or via plug-in USB/xSD devices.

Router Stack options

4. Linux based stacks

In general, any router stack based on Linux is appropriate to use when considering the addition of new software and services on to a gateway platform. RDK-B/-M, OpenWrt, and proprietary stacks are all candidates for this. In most cases an abstraction or high-level API interface is really important to be able to offer a target layer for 3rd party software to work on top of.

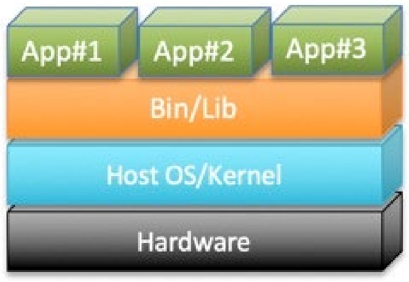

Figure 11 - HW/OS/Application Layering

4.1. RDK

RDK-B/-M itself has an internal CCSP bus (based on DBus) that acts as the backbone of the system, connecting the core RDK-B subsystems together with protocol adapters and software components. New software can be added into this system and have complete first class access to the inner workings of the platform. Support for low level interfaces, such as Wi-Fi HAL, or Cable Modem HAL, or Ethernet/Switch HAL (utopia) is also included, as is support for managing configuration settings/NVRAM. External protocol adapters for TR-069, SNMP and the Comcast developed WebPA interface allow management access to the system.

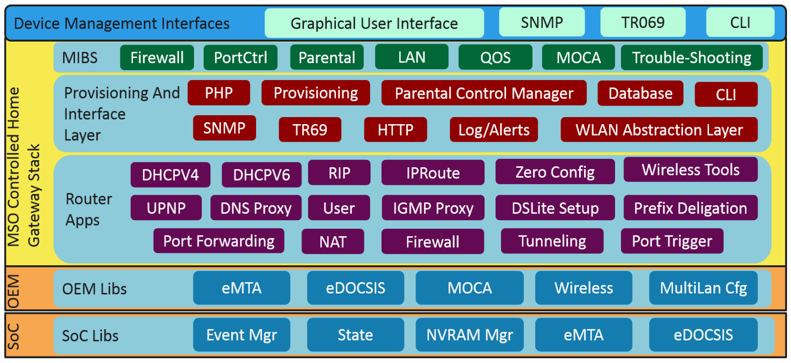

Figure 12 - Key RDK-B Software Layers and Components

4.2. OpenWrt

OpenWrt also has an internal bus, uBus, that acts as its backbone for enabling communication and control between all the internal elements that are used for routing and management. It uses “uci” for its configuration management, and offers a lot of equivalent services that RDK-B/-M offers that are typically expected in a gateway stack. prplWrt packages together some new carrier class components into openWrt.

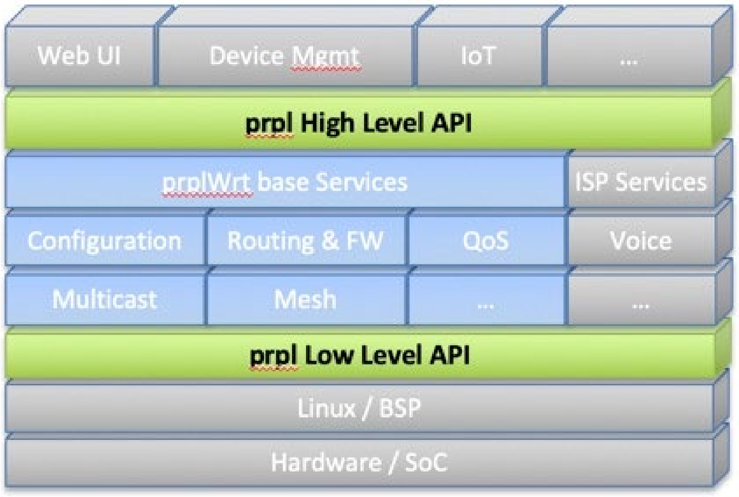

Figure 13 - prplWrt Organisation

4.3. Proprietary

Proprietary or OEM stacks, such as ARRIS Touchstone or ARRIS 9.x, all offer the same type of functionality as RDK-B/M and OpenWrt. Each stack breaks down the control functions required for each of the underlying subsystems to implement the various software and protocol requirements for a gateway.

5. SDKs, HW integration and HALs

In the main, all of these stacks are ported to run on different SOCs through the use of supplied SDKs that provide the base Linux kernel support. The SDKs mostly use Linux defined interfaces, particularly low level interfaces, when possible. When integrating extra hardware with a SOC, new drivers are provided by the 3rd party hardware supplier. In an effort to simplify the adoption of different hardware in to the router stack, say in the case of Wi-Fi, RDK-B has mandated the use of a Hardware Abstraction Layer (HAL) with a view of managing and controlling each Wi-Fi system in much the same way. This requires the hardware supplier to adapt their drivers to support the HAL. In the case of OpenWrt, the approach taken is to leverage existing Linux layers for Wi-Fi, such as cfg80211 or hostapd/wpa_supplicant, and require the hardware suppliers deliver this interface, with most of them doing so.

Other subsystems in the SOC, such as the low-level packet acceleration and switching functions are harder to get standard drivers for, as each hardware supplier does things differently matching their HW architecture. However, with advances such as Open vSwitch (where switching is performed in software), Switch Abstraction Interface (SAI) and "switchdev" it is possible to tie in these low-level hardware features in an abstract and performant way into the chosen router stack (as long as the SOC provider supports these features!).

6. Higher and Lower Layer SW interfaces

Given the advances in the hardware integration efforts, it could be assumed that the higher layer software interfaces are just as advanced. Unfortunately, this is not the case, although a lot of work is ongoing in this space. In most gateway cases, there was no real need to expose "standard" software interfaces, as no one apart from the OEM vendor was developing software for the platform. The accepted interface into a gateway was typically the network management layer, namely SNMP or TR-069, or alternatively a local HTML/web interface.

The various stacks mentioned all have internal buses for connecting their various HALs and adapters/components together. One straightforward way of exposing software interfaces to 3rd party software is to simply provide access to the internal bus. In a number of instances (say high performance network interfacing software), this is exactly how 3rd party software is integrated, using the internal bus as well as tight integration with low level driver interfaces. Such integrations can be challenging (requiring legal agreements for source code sharing, engineering access/etc.) and because two or more codebases become so tightly coupled, the only option of releasing bug fixes or enhancements is to release a completely new firmware load (going against the need for speedy releases). Such tight integration may also require more software development resources to achieve the final deliverables.

Using this model for delivering the majority of new software and services cannot scale. Such a model would also threaten the security and robustness of the stack itself, something to be avoided. What is needed are a set of defined interfaces that can be supplied to 3rd party/Independent Software Vendors (ISV) to allow them work somewhat independently of the detailed underpinnings of the firmware stack, and they will still likely need the platform tool chain to enable them build software. The following sections outline the different APIs that are available with PRPL, OVS/OVSDB, NFLua and an internal CommScope API. These interfaces are not only critical for so-called "native" software integration where software is built into the monolithic firmware image but also critical for container based options. OpenSync is also described and offers an OVSDB interface that allows a hybrid model of native code developed for the gateway that also interfaces with a remote/cloud system that may be running additional cloud applications.

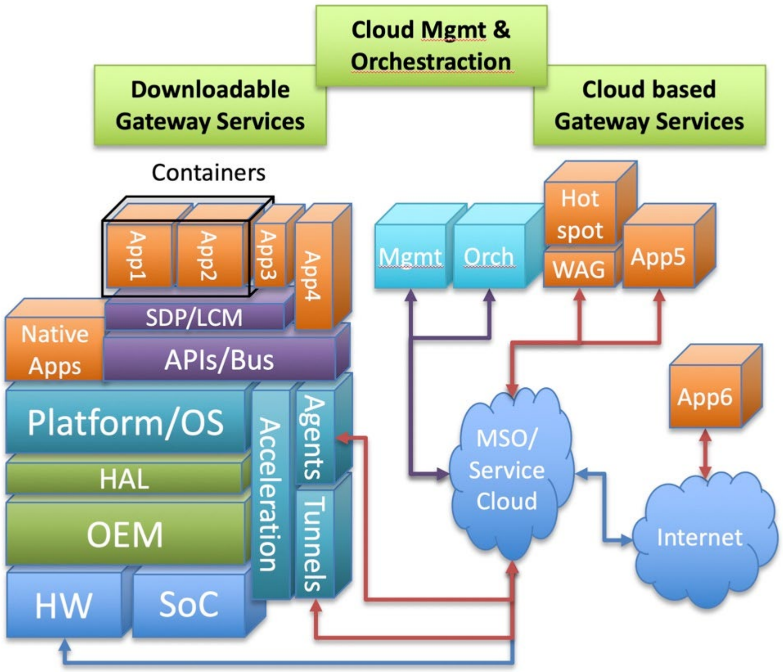

Figure 14 - Future Router Stack Architecture

6.1. prpl Higher Layer and Lower Layer API (HL/LL-API)

CommScope knows that some MSOs have been very successful with OpenWrt and would like to continue to use this stack into the future. The flexibility of OpenWrt provides for a complete stack covering most features needed for some MSOs. The prpl Foundation are also working on a carrier-grade definition for OpenWrt called prplWrt, as well as other work trying to standardize on higher level (HL) APIs and low-level (LL) APIs (e.g. cfg80211), and pushing Wi-Fi silicon vendors to standardize on the use of Linux Wi-Fi control layers to avoid proprietary drivers.

The prpl High Level API has been considered from the ground up as a platform abstraction layer to enable the delivery of new services to be easily integrated to GWdevices. The HL-API consists of a definition of 30+ primary features typically used in a GW as well as a model on how this can be integrated into multiple industry stacks, including OpenWrt and RDK-B. CommScope is currently reviewing the use of the HL-API on RDK-B, and what it will take to work over D-Bus*. The HL-API is not limited to higher layer services being added to the device, it also supports the idea of new underlying system components being added to a platform that can increase system functionality (and having this available to other software layers). The HL-API and prplAdapter also support features critical to enable 3rd party software to be added to platforms, particularly in the areas of access control and "user management". These areas are fundamental to enabling and restricting what elements of the gateway platform can be interacted with or controlled by software services.

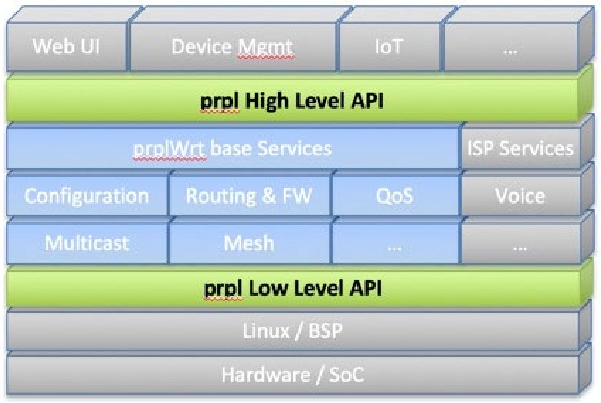

Figure 15 - prpl High and Low Level APIs

A key part of prplWrt and the higher-level APIs is to provide a so-called "prplAdapter" component that provides access as well as access-control to the inner operation of the routing/platform stack. This interface approach is meant to help the development of services required by operators, as well as exposing certain APIs to 3rd party application developers. Even though prpl has focused on OpenWrt, the major effort on the higher- and lower-level APIs is considered stack agnostic, and the expectation is that these interfaces will be available on RDK-B and other router stacks. CommScope is currently involved in an exercise to identify the work effort for mapping prpl High-level API to RDK-B, while prpl is also pushing the use of certain APIs into the RDK-B community for Wi-Fi management.

One example of software using both the prpl HL-API and low-level API is the prplMesh implementation, where a software platform exposes control of the EasyMesh controller using the HL-API while also using the prpl LL-API (namely cfg80211) for the EasyMesh agent, interacting with the low-level control and management functions of the Wi-Fi chipset. The portable prplMesh implementation for EasyMesh will run on any platform that supports the LL and HL prpl APIs, as well as exposing the necessary interfaces to allow 3rd party Wi-Fi optimization systems interact with the prplMesh EasyMesh controller function.

The set of APIs provide a significant abstraction layer to support development of both applications on the system, as well as exposing stack information (including status and monitoring information) to remote management platforms.

6.2. Open vSwitch/OVSB

New "OpenSync" software has been developed that relies upon OVS and OVSDB to expose internal operation of a gateway to a remote cloud controller. The software currently supports Wi-Fi management and has extensions for local tunneling. Current implementations use a MQTT service for actively monitoring the status of the home Wi-Fi environment back to the proprietary cloud. Open vSwitch is also a key part of this architecture, where the majority of its configuration and monitoring system had been developed. OVSDB is used in conjunction with Open vSwitch to provide a distributed database solution managed from the backoffice that controls pretty much all of the functionality of the OpenSync home deployment. The model is quite distinct from the existing/ traditional network management model used by operators be it TR-069/098/-181 or SNMP.

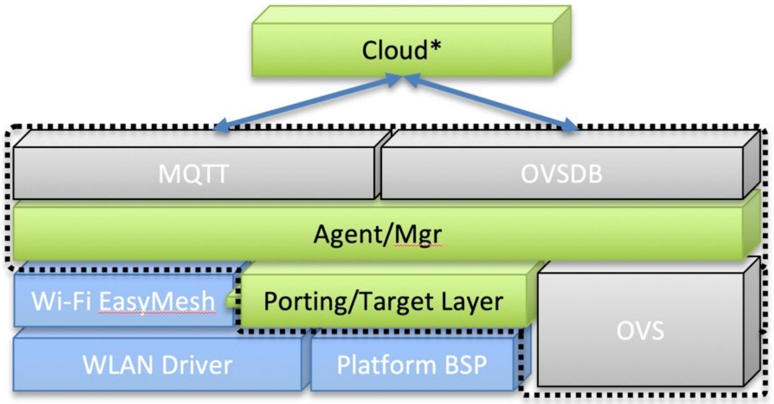

Figure 16 - OVSDB/OVS Based Architecture

The OpenSync solution is currently limited to Wi-Fi (on supported platforms) and some visibility into L2 switching. Some additional features such as basic device identification, basic speed test, and QoS control exist along with tunneling support of home network traffic between proprietary Plume Wi-Fi PODs back to the home gateway. The use of OVS for complete switch management is being considered on multiple platforms. Retrofitting it on older SoC platforms may have some challenges due to existing SoC supplied slow-path/fast-path handling and having to deal with very specific WAN access handling. However, where it has been ported, there is an option of dealing with everything relating to packet handling directly in software in the Linux kernel.

The use of Open vSwitch/OVSDB in the OpenSync has the potential to bring an SDN control plane to the operator subscriber network, and could in theory be coupled with hybrid cloud applications where traffic is selected in the home, and delivered using GRE tunnels to cloud applications that provide various software and networking functions, similar to how a WAG works today, but dealing with much more than just Wi-Fi hotspot related traffic.

6.3. Lua Based Architecture

A Netfilter/Lua based architecture has also gained some traction in the industry, with its primary software layers combining netfilter and lua extensions within the kernel to simplify the interception and handing over of packet to user space within a Linux platform. The architecture enables a scalable approach for higher layer applications for interacting with packet flows passing through the gateway.

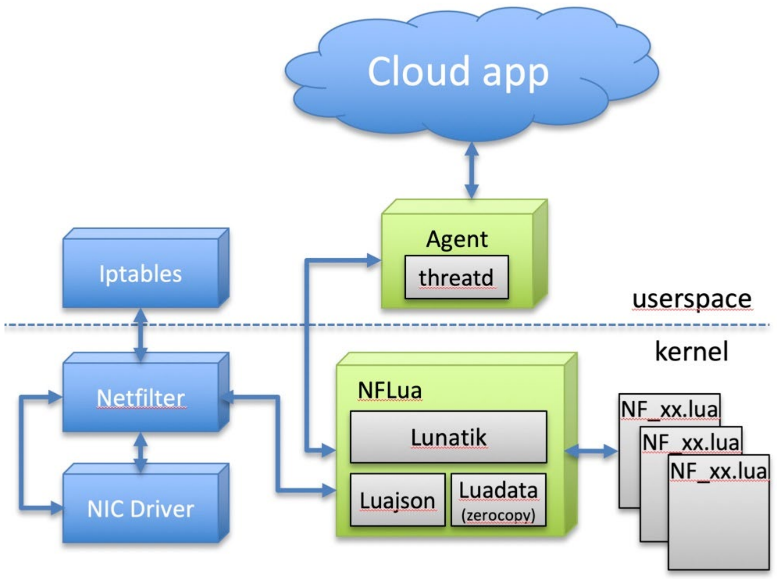

Figure 17 - NFLua Packet interception and Agent Architecture

The solution is a potential alternative to OVS. The NFLua kernel component integrates with the existing Linux netfilter and iptables for packet inspection. The model enables userspace agents to interact directly with NFLua packets that are intercepted, with the ability to operate on these packets locally (using various protocol plugins for different protocols) or to act as an agent to a cloud entity that can process these packets remotely, possibly using more complex or capable functionality not possible in the gateway footprint.

6.4. CommScope Container API

The CommScope container interface is based on providing a controlled API for ISVs to use that enables access to elements of the internal router bus. The APIs expose objects that software can use, while an access control system marshals which software can access what objects. The API interface relies on a form of object based loosely on TR-181 data definitions as well as extensions for interacting with low-level layers within the stack. The interface has already been used for several container applications.

6.5. Life Cycle Management(LCM) / Service Delivery Platform (SDP)

In addition to the need for the aforementioned interfaces both data plane and control plane for implementing software, there is also a need for software interfaces or a subsystem to manage these new software components that can operate on gateway platforms. This is somewhat equivalent to what Docker provides to manage interactions on a platform as well as interacting with a remote Docker repository hosting available applications, but one key difference is the target platform, in this case embedded gateway platforms.

Typically, Docker is used on extremely capable hardware platforms with plenty of RAM and Flash as well as large CPU resources, something quite different to embedded platforms like broadband gateways. As such, companies have been investigating more lightweight options to achieve equivalent function for gateways. Broadband Forum developed the TR-157 approach many years ago, including a key element known as Software Module Management (SMM). The SMM system provides the basis for a new Life

Cycle Management (LMC) and Service Delivery Platform (SDP) that Vodafone has created over the last number of years.

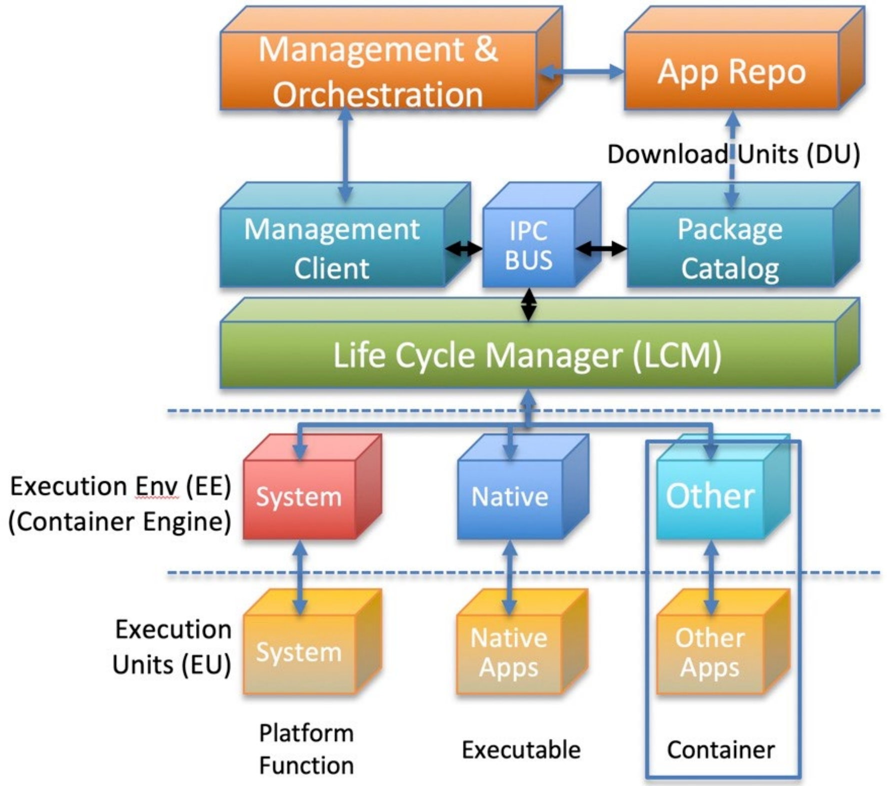

Figure 18 - prpl Service Delivery Platform/Life Cycle Management

This system deals with the full lifecycle management of software components, from arranging the download, to the provisioning and running/monitoring, and eventual removal of the software within a gateway platform. LXC is used for application containers. The solution operates with existing ACS's relying on TR-069 (or in the future USP), requiring some additional capability in the ACS to help with orchestration of what SW components are position on what gateways, etc. The system provides a complete solution and is planned to be open sourced into the prpl Foundation, providing an option that can be ported to any router platform for managing software components in a consistent way. Other companies have also been working on similar approaches and the hope within the community is that we can bring multiple parties together to create a common solution.

Container Usage - LXC or Docker/Balena

The main reason to look at containerized applications is to try and provide abstraction from the main monolithic firmware image. As mentioned, the release cadence of the main firmware images maybe too slow compared to new software feature needs. Having the ability to develop and test software that can then be deployed on top of an existing image can resolve this cadence issue. It also allows for such software to be tested against a fixed target release in the field, ensuring more confidence in the new feature when they are deployed in the field. Having independence from the monolithic image also means it's possible to upgrade such software quickly in the field in the event of issues arising, without having to perform a complete system test of the main monolithic firmware image again.

7. Container Choice

The choice of container system for gateway platforms needs to consider the overall Service Provide and Supplier model being used (at arms-length or tightly integrated), the available RAM/flash, and the long-term maintenance requirements. LXC, Docker, and Balena offer a nicely packaged system to manage applications, and in some cases also handle the application repository aspects as well. One other option that avoids application containers relies on the original Linux primitives that can limit resource used by applications/processes running in a system using chroot() and cgroups/namespaces. This alternative approach works as well as containers, and some operators are considering this more lightweight approach for managing applications in order to reduce the overhead associated with the other options.

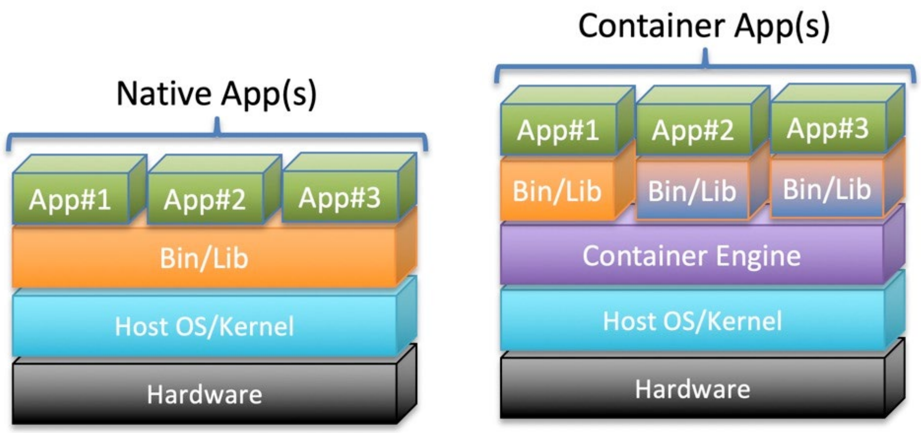

Figure 19 - Native Apps vs Container Apps

The choice of container option will likely have an impact on the build system for the devices being used. In terms of LXC containers and cgroups/namespace approach, it is possible to get much smaller container images as a result of reusing the available dynamic libraries within the primary firmware image root filesystem. A challenge with this approach however is the tight coupling required as a result of having to build the LXC container applications as part of the overall firmware image process. When working with internal SW teams, this is not a major issue, but there may be the usual "sharing problems" if 3rd party software companies need access to this build system.

Challenges such as the overall version of firmware image and versions of libraries contained in the root filesystem may change due to upgrades, fixes, new features, etc. and any LXC container application may be incompatible with the changes results in the need to have very careful feature and change planning in order to avoid a permanent state of development.

In addition to resource management and resource limits for new software and services, a key requirement to consider is how to interface with the main routing platform. In some cases, the integration requirements for new software can be limited to an IP and TCP/UDP port mapping, whereas other integrations need to directly interact with the local platform. Clearly defined interfaces (like all those described earlier) are a **must** to ensure coordinated access to the platform is maintained. Such interfaces enable 3rd party software providers understand how to interact with the platform, while the same interfaces provide a defined bridge that the platform software can marshal in terms of access control rights (what application can interact with what subsystem), and abstraction (allowing underlying systems to be modified while maintaining consistent northbound interface).

In terms of Docker or Balena containers, as they create their own root filesystems the resultant container sizes can be much larger compared to LXC. However, 3rd party developers need far less access to the internal build system or the firmware internal libraries, relying mostly on the target toolchain to build their applications. The approach enables independence between such containers and the system platform, but at the cost of RAM and flash resources.

In most cases the target platform for containerisation will require at least Linux 3.18, and preferably the latest kernel available

The choice of container comes down to the following options
- LXC Container
- Docker Container
- Balena Container

7.1. LXC Container

LXC Containers provide the closest coupling and reuse of existing resources in a GW platform, and as such are definitely being considered on platforms that need some flexibility to deal with very constrained environments. They are the pre-cursor to nearly all other container systems, being a packaging up of the previously developed kernel tools developed to "contain" processes. They tend to be light weight in terms of RAM and flash, and fit into an existing platform without much overhead in terms of "container execution environments". However, these benefits come at a cost of tight integration and reliance on a flexible firmware build system. LXC also does not have a native "orchestration" option that can be used directly, resulting in the need to create a suitable environment (the previously mentioned LCM/SDP addresses this need).

Multiple industry efforts are underway to add LXC containers to embedded GW platforms, with options being discussed with most of the Tier-1 operators.

7.2. Docker Container

Docker is an open platform for developing, shipping, and running applications. Docker enables the separation of applications from a local platform to enable speedy software delivery. Docker execution environment provides isolation and security allowing multiple containers to simultaneously on a platform. Compared to virtual machines, Docker containers are much lighter, but these containers and the execution environment can be a lot heavier in resource usage compared to LXC.

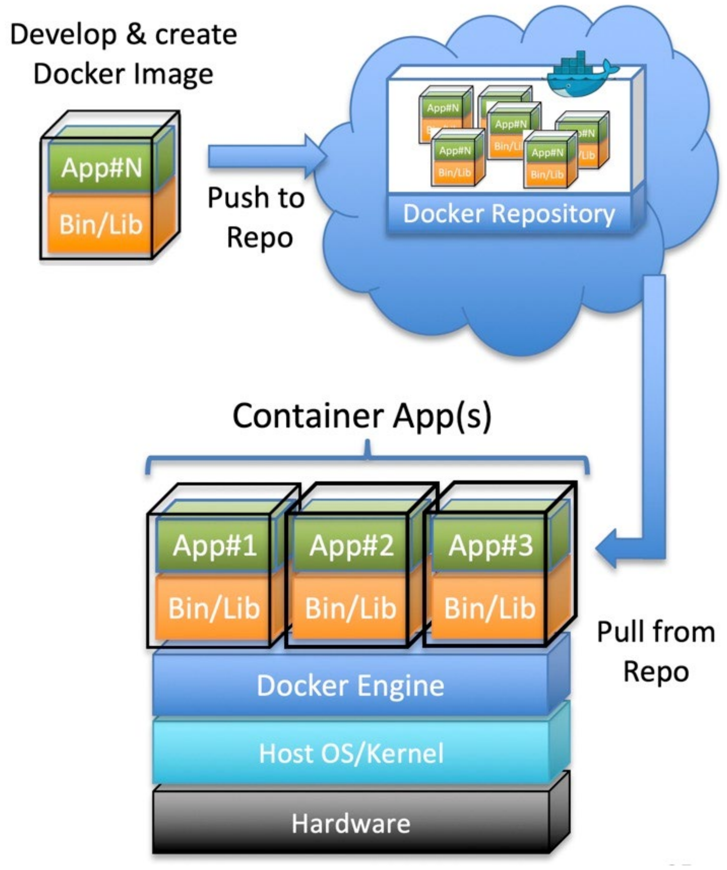

Figure 20 - Docker Ecosystem

A major benefit of Docker is how it creates containers, where every file/library/application required is packaged into a single container image enabling it to be distributed in a highly portable fashion. As a result a Docker container can be deployed on a gateway, local laptop, physical, or virtual machine in a datacenter or in a cloud provider environment. The portability of the Docker container means that many more software providers can develop their applications to run on Docker, enabling a very rich and vibrant market space. A Docker container is a runnable instance of an application image. Like LXC it can be started and stopped using a Docker API or a CLI. Docker relies on Linux services (either natively in a Linux kernel, or through "Linuxkit") and uses namespaces in the same way as LXC does to provide the required workspace isolation for the container to operate within. Namespaces offer process, networking, inter-process communication, mount/filesystem, and some kernel isolation.

In addition to these fundamental features, Docker introduces a whole host of extra functionality to be able to manage and interact with containers, enabling eco-systems to be built to fully manage and orchestrate the operation of large numbers of Docker images/containers over vast "fleets" of compute resources.

7.3. Balena Container

Balena containers are very similar to Docker, having been developed as a cut-down version of Docker. The following features have been removed from Docker Container support to create the lightweight Balena container platform, resulting in a 3.5x reduction in size:

- Docker Swarm

- Cloud logging drivers
- Plugin support
- Overlay networking drivers
- Non-boltdb discovery backends (consul, zookeeper, etcd, etc.)

Balena concentrates on using RAM and storage more conservatively and focuses on atomicity and durability of container pulling. These facets are ideal in the context of embedded systems, compared to the more traditional cloud systems that Docker is targeted at).

Container Experience in Gateways

8. Containers on Commscope Gateway Platform

The Docker ecosystem is being used on some CommScope gateway devices, with a view of enabling common applications to be deployed across a range of mixed capability devices (concerning CPU, RAM and flash resources). The current management/orchestration of the Docker system on these platforms relies on the TR-157 (SMM) functionality previously mentioned as well as more explicit Docker controls. The SMM system has some roots in the Home Gateway Initiative NERG as well as previous efforts that tried to add OSGi to gateways. The TR-157 defines platform attributes as well as lifecycle management.

The CommScope gateway platform relies on the open source Docker Engine to provide the framework for hosting containers. A Docker Client is added to the gateway to manage and control access to the Docker environment. Remote Docker clients are also supported to assist with the installation of containers as well as querying status/etc.

SMM depends on Execution Environments (EE), Deployment Units (DU) and Execution Units (EU). The Docker Engine is equivalent to the EE, providing a platform for hosting applications that are effectively sandboxed to the rest of the gateway/host system. Docker Images are equivalent to the DU, providing a way of managing the specific files/etc. associated with the application being downloaded. The EU is the active running Docker Container executing within the Docker Engine/EE environment.

Docker containers are either pre-downloaded or downloaded from the Docker Registry. Interactions with the Docker Registry, including authenticating access, are all logged to ensure diagnostic information can be reviewed in the event of issues.

The current model is to use the CommScope Container API for Docker Container applications control objects on the gateway platform itself. Extensions such as providing access control to local Linux services and Dbus access are also provided. From an operational perspective, as some platforms are flash limited, the Docker Engine itself is run time installed into RAM, as are the other Docker Container images.

The running of the Docker Engine on the platform requires allocation of resources from the gateway for any Docker Applications being deployed. The current support in the gateway based Docker support is for installing, enabling, uninstalling, and disabling using either an External Docker Client or using the Docker Configuration file. The main features of the SMM are provided to report on status/etc. of the Docker Engine and running applications.

Some of the Docker applications include McAfee security gateway as well as SamKnows. Other applications are considered as well as internally developed features.

9. Life Cycle Management (LCM) on OpenWrt platform

A previously mentioned the LCM/SDP platform developed by Vodafone enables the management of downloadable software components in LXC containers. This was demonstrated through a POC that Vodafone developed and demonstrated at openWrt 2018.

This POC chose to use the primary features of the TR-157 Software Module Management (SMM) specification for this, providing a generic interface for this interface, allowing it to be mapped into TR-069 for ACS management or made available for other agents to use with other orchestrators. The LCM component provided external access to execute the available Life Cycle Management API methods, while also being responsible for fetching packages/containers, retrieval of information about packages from the local filesystem, as well as delivering the required applications to the Execution Environment to run.

The POC demonstrated the use of multiple Execution Environments (EE) allowing for mixed service operation. A Base EE was used to allow upgrading of specific components into the main root filesystem, that did not require any separation, such as new native images. A key feature of the Base EE was to allow direct patching of the main OpenWrt system, enabling the installation of a new native package directly into the running system. The use of the Base EE also allowed for a bit more package information to be included to be able to authenticate the packages, etc. A so-called Native EE was added to enable root filesystem separation, meaning that a new Native package would not overwrite anything in the base root filesystem, enabling isolation from the running system. The final EE was the Container EE, where new 3rd party applications needed isolation from the main system, and would be have limits placed upon all resource usage, as well as preserve system stability.

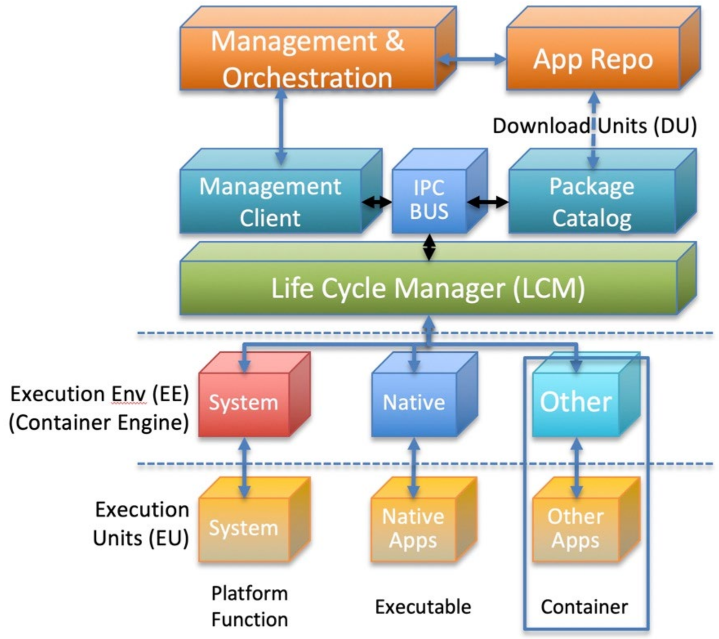

Figure 21 - SDP/LCM and Orchestration Overview

In all cases the LCM was responsible for managing the different EE, where it would perform actions on the EE, and deal with the returned status. Operations such as install, uninstall, start and stop were all supported.

The following diagram shows the model used in the POC:

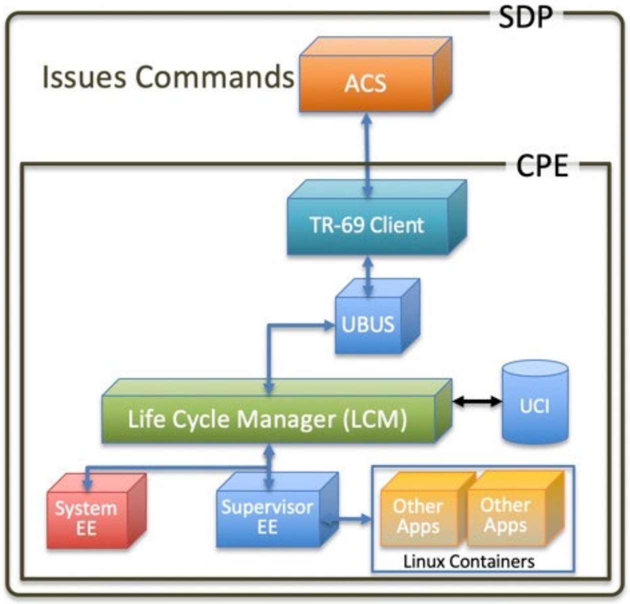

Figure 22 - SDP/LCM POC Configuration

The ACS platform was used to issue commands to the CPE platform, where they were handed over to the LCM to perform all actions related to running services within the platform. The System EE listed above allowed for the ACS to request package updates (OpenWrt) to be applied to the running system, while the Supervisor EE was used to actually run the isolated applications. The Supervisor EE is responsible for handling the environment that applications run with

The Supervisor EE supported features such as package verification and install/remove, service startup/shutdown, as well as isolation (including limiting namespace, RAM and CPU). The OpenWrt Summit 2018 demonstration showed some basic containers running, as well as a more complex setup that involved Samsung SmartThings integrated in a container, downloaded into the system and using a local Zigbee USB dongle to interact with an external Zigbee lightbulb. Other aspects such as CPU resource limitation were also demonstrated. All of the interactions in the demonstration were controlled using the SMM functionality on the connected ACS.

The SDP/LCM system as currently defined delivers a complete solution for managing containers and even native applications on embedded gateways. It offers orchestration through the connected ACS (although ACS platforms probably require custom extensions to really hope to act as orchestration systems), and works on OpenWrt. Work is ongoing to get this functionality working on RDK and hopefully the overall SDP/LCM solution software will be opensourced at some stage.

10. Nomad POC

Nomad (from Hashicorp) is a highly scalable orchestration system that has been deployed to deal with launching 100,000s of container applications for various purposes. It manages clusters of machines and runs different types of applications on top of them, integrating with another Hashicorp product called Consul (a service discovery and configuration tool). Its primary function is to manage microservices efficiently over clusters. A Nomad POC was developed to demonstrate the management of Docker based containers being orchestrated using the Nomad system (www.nomadproject.io). This POC relied on integrating Docker CE onto a platform, running a Nomad Agent container on the platform and using Nomad server to interact with the agent to orchestrate the setting up of "IOT" application container and some other basic containers.

The work began on a platform with 512MB RAM and 4GB flash. The POC team already had extensive experience with Nomad Server and were using this to understand the client side and how this would scale. The Nomad Agent (that runs on a client device, such as a gateway) includes support for so-called "Task Drivers", allowing it to manage multiple types of execution environments, including, Docker, Isolated Fork/Exec, Raw Fork/Exec, LXC, Java, QEMU, Rkt, Custom. For the purpose of the POC, Docker was the chosen environment.

The POC demonstrated that the platform was well capable of delivering the required services. The size of available flash used was considerably more than on most embedded platforms today, where for example a gateway might only be designed with 128MB flash, which is only **3.2%** of that available in the POC platform. The POC was capable of demonstrating the use of Nomad for orchestrating the local services, as well as showing that an IOT container application was able to function and access a local Zigbee interface on the gateway.

Alternative Virtualization Options

Containers are not the only virtualization option that exists. The following sections explore virtual machines as well as full/hybrid cloud applications.

11. Virtual Machines

Virtual Machines (VM) are another way of isolating software functionality to run on a platform. They require a complete running platform of software, including a complete OS. When a VM runs on a platform there are different ways it can be position. It can run on top of a so-called "baremetal" hypervisor (the software to manage VM access to the local HW platform, also known as a 'type-1' hypervisor). Systems such as VMware ESXI, or Microsoft Hyper-V server or open source KVM are all examples of a type-1 hypervisor. Alternatively, a VM can run on top of a 'type-2' hypervisor running in an OS on the HW platform. VMware Workstation, Oracle VM VirtualBox, and Parallels are all type-2 hypervisors. VMs provide a complete environment, meaning that in most cases they require massive amounts of RAM and storage to operate, one example of a ubuntu 18.04 desktop VM requiring 25GB of storage and 2GB of RAM. As such, VMs really don't suit when trying to add some basic software features on top of an existing embedded system.

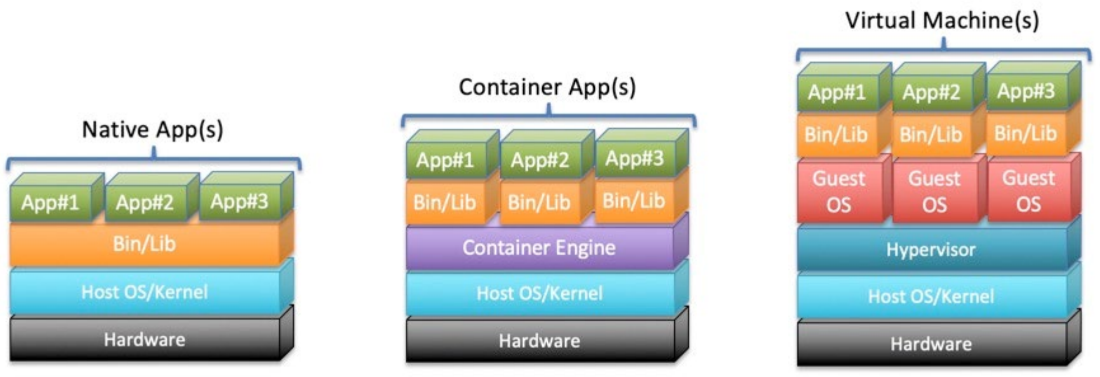

Figure 23 - Native, Containes and Virtual Machines

VMs are not discussed any further in this paper.

12. Full and Hybrid Cloud Virtualisation options

The traditional model of Virtual CPE has talked about moving the entire routing platform software out of the home and into the cloud, and having all of the network traffic hairpin through this remote virtualization platform that replicated all of the home networking functions. The hope of vCPE was to reduce the cost of the gateway device and move the SW complexity to the cloud. The approach was to enable different SW instances in the cloud that would support centralized feature development and versions, with the ability to rapidly cutover gateway devices from one version of software to another to get new features, as well as to offer new software features and services to customers, regardless of the type of gateway hardware was in each home.

This model has not really succeeded. The costs of offering the vCPE platform in the cloud while also offering gateway HW in the home never quite added up to something that was more economic than a dedicated gateway in the home. The truth is that vCPE hardware platforms, especially with Wi-Fi, are not too different in costs compared to equivalent home gateway platforms, in most cases the actual SOC is the same and offers the same MIPS processing power. The main difference would be RAM/flash costs, with a vCPE platform requiring less of both (but ironically, could be forced to buy more than required just to hit RAM/Flash price sweet spot).

So, is vCPE dead? The answer is no, as some very good pieces of vCPE can be used. The idea of isolating certain traffic flows and certain Virtual Network Function (VNF) software to the cloud is an idea that has persisted and been demonstrated to work well. In this case a traffic tunnel connects the home gateway to the remote cloud VNF, where all the hard work is performed. One of the main examples of this is “Wi-Fi Public HotSpot” services. The traffic to be tunneled is simply that traffic that operates on one of the Wi-Fi SSID that the gateway offers. Every data packet is received from the SSID and tunneled using a ‘softGRE’ tunnel to the cloud VNF. The cloud VNF terminates the tunnel, extracts the traffic and operates a Wireless Access Gateway (WAG) function, that deals with AAA and all the required traffic management (DHCP, etc.) and encapsulation/decapsulation, before dispatching the traffic off to the internet. This model is one of the first real examples of vCPE and has been widely adopted.

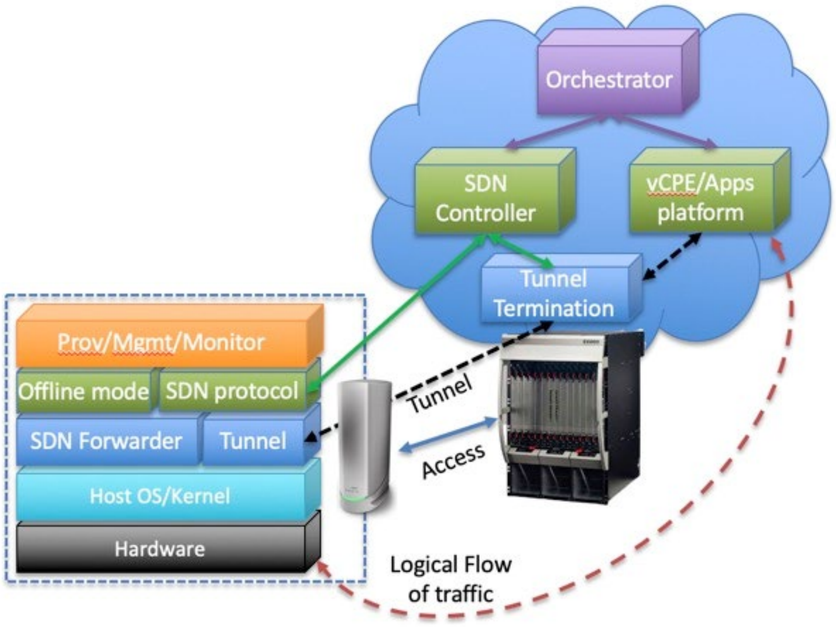

Figure 24 - vCPE with Cloud Services

However, it's a very basic option, using a course traffic filter (the entire SSID) to isolate traffic. The main function the gateway must provide is the ability to isolate such traffic and pack it into a SoftGRE tunnel connected to the cloud VNF, so it's definitely minimized SW complexity in the gateway.

More advanced versions of vCPE have started to be developed, using more fine-grained data plane filtering options. In a lot of cases, traffic that is filtered must be transported out to a remote cloud VNF where the actual software processing occurs, typically through a SoftGRE or equivalent tunnel. Alternative options also exist where this traffic could be handed over to a local container or software component, mixing up the different models (where is makes sense).

Data Plane

Traditionally, Linux network tools, such as iptables have been used to manipulate traffic flows, providing low level filtering and redirection/etc. These tools are used by some of the key networking functions within the routing platform, but typically are not open to higher layer software components, as they have the potential (if used incorrectly) of wrecking the network packet forwarding of a system. No real programmatic API has been developed to expose this interface to 3rd parties. However, Software Defined Networking (SDN) does offer some new ways around this.

The basic tools of SDN, such as openFlow and Open vSwitch, have offered the ability to isolate incoming traffic flows on a platform and modify or redirect such flows for additional software processing, including forcing a flow to be sent out an interface that happens to be a tunnel or another local interface, possibly connected to a container. New software approaches for gateways are starting to reuse this type of processing.

The benefit of this model is that once the software agent is enabled on the gateway platform, then any interesting traffic flows can be dispatched via a tunnel interface to a remote cloud VNF, without requiring new software to be added to the gateway (filtering instructions would simply be configured into the gateway depending on what traffic flows had to be isolated).

Integrating the opensource OVSDB and Open vSwitch (OVS) into a gateway has enabled OpenSync to exert very fine-grained control over traffic passing through a platform, with the possibility of redirecting such traffic to a tunnel interface for carriage to a cloud VNF. The benefit of open source OVS is that it is possible for 3rd party software to also use the same infrastructure if required.

A similar packet interception model that embedds a NFLua component linking to the Linux Kernel network packet handling has also been developed. This has been used to provide sophisticated AI driven cybersecurity and network intelligence features for network operators. The ability to deploy an agent and then dynamically reconfigure its basic rules provides a very powerful model that allows for independent upgrades/etc. without having to involve an operator at all. Such an agent module could also be repurposed to provide a packet filtering option, like OpenFlow, to redirect traffic to a remote cloud VNF.

In these traffic interception/filtering/redirection cases, the traffic is either hair-pinned out to the cloud VNF and sent back to the gateway, is completely consumed by the remote service, or dispatched to a local agent present in the gateway that also performs processing or other software handling. Using these tools enables easier manipulation of the data plane than ever before and offers more organized control about how to isolated traffic and direct to software components (local or remote). More effort is being put in by SoC providers to ensure that hardware acceleration can also be applied to this traffic manipulation, ensuring that software can access the high speeds expected from gateways.

Control Plane

As mentioned a lot of container systems have their own proprietary backends for controlling how containers are deployed and operated on compute platforms (e.g. gateways in the case). These tools are more concerned with treating the containers as black boxes and satisfying the “label” of resource requirements that come with the container.

In the case of the ARRIS Docker Container POC, additional supports were provided to allow the manipulation of the Docker system from a remote ACS by using TR-069 extensions mapped into the TR-157 SMM system, allowing some more native (from an operator perspective) management to be employed. Kubernetes was not used to provide orchestration in this instance.

The SDP/LCM system that Vodafone has created also uses a similar model to ARRIS, relying on TR-157 EE, EU and DUs to enable a very flexible control system for managing sophisticated software delivery options and life cycle management. This system also relies on the use of TR-069 to assist with orchestration/etc.

Nomad is another orchestration system capable of flexibly managing many different images (via nomad agent). It is capable of dealing with Balena, Docker and LXC container images, as well as many other image options. Like Kubernetes, Nomad can scale very well in a data center setting, coping with very high container deployment scenarios. However, Nomad and Kubernetes may not be able to scale to the required number of containers when deployed in an operator environment with thousands or millions of devices with multiple containers per device.

Existing ACS platforms may be able to cope with the scale of unique devices, but need additional “orchestration” extensions to be added to them. ACS platforms already deal with firmware image management/etc., and the TR-157 SMM option extensions provide a defined model for managing the EE, EU, and DU options in a gateway.

Concurrency and Orchestration Scalability

Orchestration must deal with a number of constraints within the embedded gateway ecosystem when used to manage the deployment of multiple applications in containers across the footprint. For one, any orchestration system must be capable of dealing with the massive numbers of deployed devices. It must also be able to cope with the potential of many different applications and application types per gateway, as well as different gateway types (varying in SoC supplier, CPU, RAM, and flash at a minimum).This just cries out for sophisticated orchestration systems that can address the multi-dimensional complexity.

What currently is not understood is the level of concurrency of applications running within gateways. What this means is whether the limited resources in a gateway are going to be under pressure if multiple applications are deployed, and if some clever orchestration technique will be required to constantly add and remove applications on demand or on a timed basis.

What is also not understood today is if operators will only allow their own curated container based software and services to run on these gateway platforms, or will decide to open up and potentially monetize the platform, allowing 3rd party applications to run, similar to the Android Play Store or IOS App Store. Given the high level interfaces and various access controls available with these, it does appear as a possibility, and may allow for hybrid mobile applications and other software services (such as IOT systems) to be developed that rely on an "always on presence" in the home rather than having to pay for high latency cloud based servers.

In terms of concurrency and high application counts, one of the easiest ways of addressing this is to basically ensure sufficient storage and memory is available in the platform. Such an approach means applications are rarely removed and replaced with other applications, thus avoiding a never ending game of Tetris that the orchestration system must play – constantly trying to fit apps into available space. This does at a slight cost of extra storage (the RAM can be freed up if an app is no longer active) but removes the need for a complex orchestrator.

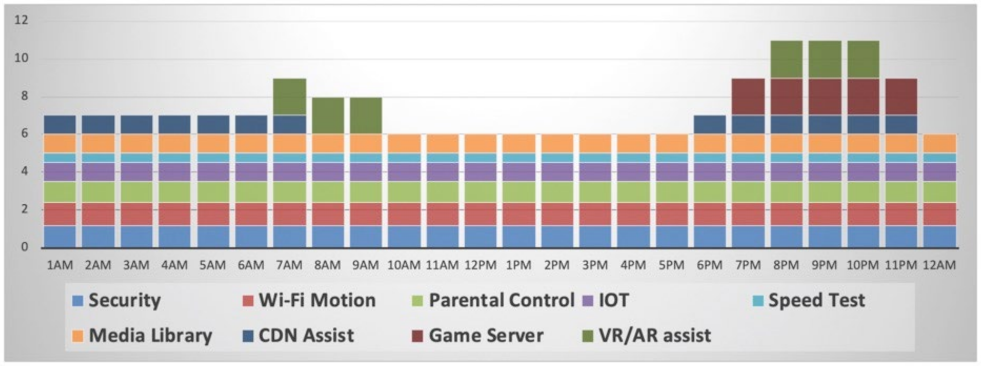

Figure 25 - Potential Sevice Load over 24hr Period

If the orchestration complexity outlined can be removed (through extra storage/etc.), then it's quite feasible to believe that the existing NMS/ACS systems that already manage vast numbers of broadband devices should be capable of supporting the required orchestration function. Existing TR-069 systems maybe usable, but the upgraded Universal Services Platform (USP/TR-369) protocol from the Broadband

Forum maybe be better suited, given the new features it brings to managing broadband devices, as well as its backward compatibility to existing data models like TR-181. USP modifies the transport protocol in use, providing a faster and more scalable link between the ACS and device populations. CommScope has recently opensourced a complete USP agent implementation that is available for integration with existing gateways to enable this new functionality.

The days of all software being delivered as a monolithic firmware image are numbered. The availability of all the required elements to create new portable software is very encouraging. The new dataplane and control plane options enable application developers (including ISVs, open source developers and the MSO community) the option of creating new applications not considered before. Along with the system high level and low-level APIs, developers are able to bundle all their required libraries and executables within a container based system (be it LXC, Docker, Balena or others), and have these orchestrated on to gateway platforms. The addition of LCM/SDP as well as reuse of Docker/Kubernetes, Nomad, or TR-069/USP based orchestration systems will enable cable operators more control over what to deploy and when/how to deploy.

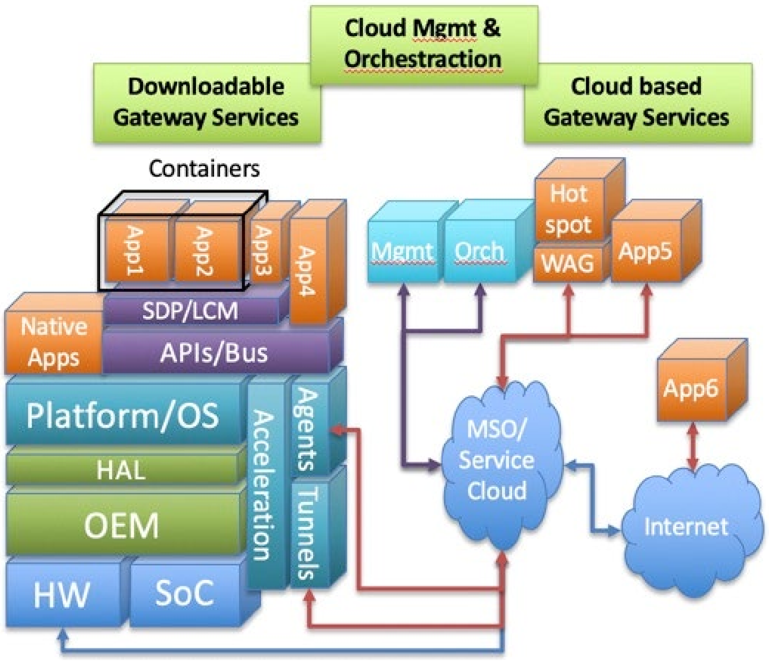

Figure 26 - Multitude of Options for Virtualised CPE

Right now there are a multitude of Docker based container applications, while only a few 3rd party container applications have been totally focused on embedded gateways. Expect this to change very soon as hardware profiles change and the various software layers and interfaces are developed and adopted in the multiple routing platforms that exist in the embedded broadband gateway world.

Conclusion

The constant change in the broadband gateway space is driving demand for newer software services at an unprecedented rate. As a result, the complex gateway platforms need to innovate faster at the software architecture level and hardware must keep in step to match the software needs.

Native application and container applications need to take advantage of new APIs, HALs and Service Delivery Platforms that are emerging to ensure fast adoption on to gateway platforms.

SDP/LCM and Docker are good options to consider for container deployments, with other platforms like Nomad also to be considered in this space. However, orchestration systems that can manage the scale of broadband gateway deployments and mixed deployed services have not been realized yet, resulting in the potential use of existing or future ACS (TR-069/USP based) to handle this workload.

Getting these new software services into gateways is essential for MSOs to entice and retain subscribers

Abbreviations

ACS	Auto Configuration Server
AP	access point
API	Application Programming Interface
AR	augmented reality
BSP	board support package
CDN	Content Delivery Network
CLI	Command Line Interface
CPE	Customer Premise Equipment
CPU	Central Processing Unit
DOCSIS	Data Over Cable Service Interface Specification
EE	execution environment
EPON	Ethernet Passive Optical Network
EU	execution unit
GPIO	General Purpose IO
GW	gateway
HAL	Hardware Abstraction Layer
ISBE	International Society of Broadband Experts
IoT	Internet of Things
LCM	life cycle management
LED	light emitting diode
LXC	linux containers
MQTT	message queuing telemetry transport
MSO	Multiple System Operator
MoCA	Multimedia over Coax Alliance
NFV	Network Function Virtualization
NFVO	NFV Orchestration
OEM	Original Equipment Manufacturer
OS	Operating System
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PRPL	prpl Foundation
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
SCTE	Society of Cable Telecommunications Engineers
SDK	Software Development Kit
SDN	Software Defined Networking
SDP	Service Delivery Platform
SOC	System on Chip
UBUS	OpenWrt micro bus
UCI	Unified Configuration Interface
USB	universal serial bus
USP	user services platform
VM	virtual machine
VR	virtual reality
WAG	Wireless Application Gateway
vCPE	Virtual CPE

Bibliography & References

Docker Information; https://www.docker.com/

RDK-B Architecture; https://wiki.rdkcentral.com/download/attachments/23593288/arch.png

Prpl Foundation; https://prplfoundation.org/working-groups/prplwrt-carrier-feed/

OpenSync; https://www.opensync.io/documentation

Safe browsing using Lua; https://www.lua.org/wshop17/Lourival.pdf

prpl Service Delivery Platform; https://openwrtsummit.files.wordpress.com/2018/11/sdp-openwrtsummit2018_1-3-1.pdf