VIRTUAL EXPERIENCE
OCTOBER 11-14

2021 Fall Technical Forum
SCTE • NCTA • CABLELABS

# A Common Remote PHY Software Stack for all RPDs?

A Technical Paper prepared for SCTE by

**Michael Robinson**
Remote PHY Software Architect
Comcast Cable Communications
1701 JFK Blvd – Philadelphia, PA 19103
+1 (404) 723-7847
michael_robinson2@comcast.com

**Jorge Salinger**
VP, Access Architecture
Comcast Cable Communications
1701 JFK Blvd – Philadelphia, PA 19103
+1 (215) 439-1721
jorge_salinger@cable.comcast.com

 

# Table of Contents

**Title** **Page Number**



## List of Figures

**Title** **Page Number**



 

# 1. Introduction

We have all heard the term "Software Fragmentation", especially in the context of mobile device applications. Software fragmentation occurs because of the variety of hardware and software platforms. As a consequence, applications may not be compatible with that new hardware, forcing the need for application changes or the need for another application altogether.

The above also happens with Remote PHY Devices (RPDs). A single cable MSO will typically deploy RPDs from multiple suppliers, each with a different application software, and even multiple RPDs from the same supplier that have different hardware components and require different application software. Although RPD software design is based on common specifications, the behavior of the software differs between RPDs implemented with different hardware components and/or by different suppliers. As these differences accumulate, managing the system and maintaining interoperability becomes more complex.

How can this be mitigated?

A common software base can be shared between RPDs, even when they come from different suppliers. This paper will explain how sharing a common software base is implemented, and how it is applied to the RPD platform and supplier ecosystem.

# 2. Typical Cable Access Network

Most MSOs' hybrid fiber-coax (HFC) networks have been designed with an upper spectral boundary of 750 or 860 MHz, while some are designed to support 1 GHz and other newer networks designed to support 1.2 GHz. For the more abundant 750 or 860 MHz networks, if not already fully utilized, it is expected that the use of their capacity will soon be increased to the point of exhaustion.

The increased utilization of this access network capacity has been driven by the success of cable MSOs' service offerings.

As it is well known and understood, for many years the growth in, and demand for, more video programming resulted in the need to allocate large numbers of EIA (Electronic Industries Association) channels for video services, mostly known for their fixed, 6 MHz size, and used both for BC (Broadcast) and NC (Narrowcast). These 6 MHz EIA channels have filled every available portion of the spectrum.

Additionally, the success of, and growth in, HSD / broadband services continues. Most, if not all, operators offer increased service tiers and observed a growth in the use of HSD service capacity for well over a decade now, which amounts to a significant year-over-year compounded growth. This phenomenal success drives the need for increased network capacity, which is implemented by either expanding the spectrum allocation for HSD, or continuous service group segmentation, to reduce the number of users per service group. Or both.

In a separate but parallel trend, operators have been actively converging video and data services into a common Converged Cable Access Platform (CCAP) platform. This evolution, which has been underway for many years now, is intended to reduce the environmental requirements in HEs (headends) that result from service group segmentation. This is because as more service groups are added through segmentation, more HE equipment is needed, which drives the need for space, power and cooling. These trends imply an evolution towards newer, more modern, and denser equipment.

However, as the success of high-speed data and on-demand services continued, the evolution of the access network progresses towards further expanded capacity and ever-smaller service groups. For the

former, the spectrum allocated to narrowcast services increases, driving operators to deploy Data Over Cable Service Interface Specification (DOCSIS©) services including more SC-QAM (Single Carrier Quadrature Amplitude Modulation) channels, and more recently, with DOCSIS 3.1, the allocation of network spectrum for more or wider OFDM (Orthogonal Frequency Division Multiplexing) channels, as well as more narrowcast video services. For the later, more CCAP ports are needed, which drives the deployment of more line cards and eventually more chassis. These expansion trends result in a continuous growth of headend equipment, which is already starting to exceed the capacity that headend facilities can support.

## 3. Benefits of DAA

The above trends are now intractably linked to two additional evolutions: distribution of components of the access network, implementing a Distributed Access Architecture (DAA), and virtualization of the core network functions.

There are many benefits from the implementation of DAA. One key benefit is the improvement on performance, which is achieved in multiple ways, including: improved SNR characteristics, enable longer link distances between the headend and the nodes, provide higher service reliability, better use of capacity, etc. Beyond the performance improvements, a key benefit of DAA is the increased headend capacity. The implementation of DAA makes it possible to improve the density of CCAP devices in several ways, including the implementation of denser equipment, the use of Ethernet technology which is simpler and smaller in footprint, which more than doubles the capacity as compared to traditional CCAP chassis.

DAA can be implemented in many ways. As the optical link from the headend to the node is converted from analog modulated forward to digital, using Ethernet as the transport, several approaches can be taken for the implementation of the remaining headend components. One approach, for which a key goal is to convert all required components into functionally individual software pieces implemented independently, is to implement DAA in various discrete SW components as shown below.

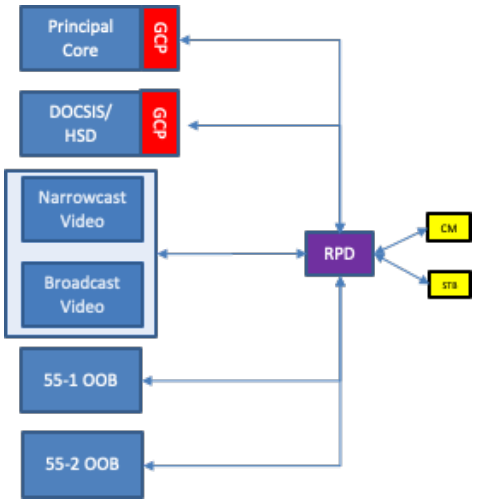


**Figure 1 – DAA Implementation Components**

The key components depicted in the DAA implementation above include:

- The Principal GCP Core, or GCPP, is the first component that the RPD will contact after receiving an IP address. GCPP is implemented such that it will configure all the RPD functions except DOCSIS channels and behavior, which will be implemented by the DOCSIS CMTS. As the name implies, GCPP communicates with the RPD using the GCP protocol. Included in the GCP Principal Core are all the non-DOCSIS command and control functions for the RPD, including configuration, management and reporting.

- The DOCSIS Core, which is the second component that the RPD will contact in the network. The DOCSIS Core also communicates with the RPD using GCP, and provides all configuration, command and control for DOCSIS channels, both downstream and upstream.

- Narrowcast and Broadcast Video engines, which can be implemented as separate components or combined into a single device, provide all the video content services for the various RPDs in the network. It is important to note that neither the Narrowcast nor the Broadcast Video engines communicate with, nor have knowledge of, the RPDs. Instead, services are configured statically in the Narrowcast and Broadcast Video engines upon their bring-up, and are multicast to all RPDs, which listen for these services as configured by the GCP Principal.

- Out-of-Band engines or cores are implemented separately from the video engines. Given that video systems are implemented using a single encryption and command/control technology, only one (i.e., either SCTE 55-1 or SCTE 55-2) of them is deployed in any one system. The OOB function may or may not implement GCP for communicating with the RPD. When GCP is implemented the OOB server is a Core, and it will configure the OOB downstream and upstream OOB channels in the RPD. However, when GCP is not implemented the OOB server is an engine, and the GCP Principal will configure the downstream and upstream OOB channels.

- Finally, not depicted in Figure 1, is a very important component: the Timing Server. Also known as the Grand Master, the Timing server provides the critical timing synchronization for all the DAA components. Each of the DAA components will include a Timing Client, which will communicate with the Timing Server to maintain timing synchronization. While timing synchronization is not absolutely critical for video services, it is imperative for DOCSIS service to operate. Therefore, video services may be initiated before timing synchronization is achieved, but DOCSIS services will not.

The key advantages for the above architecture are: implementation consisting of a multi-supplier platform where each component can be developed independently, smaller functional components with simpler implementation, and generally reduced time-to-market. However, implementation of smaller discrete components has its downsides, such as: the implied requirement to more tightly specify the behavior of each component to ensure that the overall system will operate as intended, management of the various components including their configuration and upgrade, and the need for more elaborate orchestration.

## 4. Common Issues Faced with RPD Deployments

The base implementation of a DAA system is generally simple. However, significant complexity is introduced when interoperability with different suppliers' components is introduced.

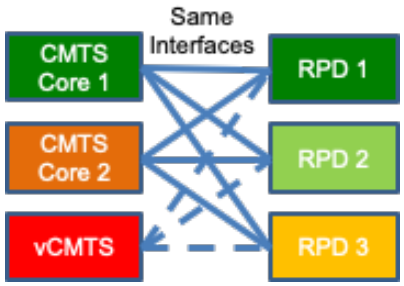


**Figure 2 – Functional CMTS-RPD Interoperability Matrix**

When considering the overall CCAP system, the complexity to achieve multi-supplier interoperability is even larger. Over the years, larger MSOs have deployed multiple CMTSs, and eventually multiple CCAPs, experiencing the complexities associated with such equipment implementation interoperability.

As depicted in Figure 2, the number of combinations of interoperable components increases geometrically as additional components are added on either side of the interoperability matrix. Having a single CMTS to interoperate with multiple suppliers' RPDs is complex and requires a lot of careful planning and implementation. But if the number of CMTS implementations is increased to 2 or 3, the interoperability complexity doubles and triples respectively. Furthermore, in the case of DAA, if multiple GCP Principals and/or multiple DOCSIS cores, and/or multiple video engines, and/or multiple OOB engines/cores are introduced into the mix, the amount of complexity and work required for lab and field testing to maintain interoperability increases by orders of magnitude.

Therefore, a multi-supplier RPD deployment coupled with a single headend implementation is a sensible approach to an interoperable DAA ecosystem. In this way, the HE components of the DAA, which are deployed in the hundreds, are kept the same for all MSO sites, but the component that is deployed in the thousands, tens of thousands, or even hundreds of thousands, are obtained from multiple sources, ensuring an innovative and competitive ecosystem.

Finally, it is important to recognize that there are numerous types of HFC nodes and network use cases, which will drive the need for an even larger variety of RPD types.

In the same way as there are different kinds of nodes for different HFC network applications, there are also Remote PHY devices with different characteristics that are best suited for each of the specific HFC network use cases. For example, nodes that are best suited for N+x network architectures may have different RF characteristics, and their corresponding RPDs may be implemented to support more service groups. By contrast, N+0 network architectures have different node RF characteristics and require RPDs that are intended for fewer subscribers.

Similarly, while there are use cases for RPDs in the outside plant, there are also applications for RPDs in headends, or "inside plant" as it is frequently called, which will have different implementation characteristics. This use case diversity has driven suppliers to offer RPDs packaged for nodes, where there is a single RPD, or in some cases 2 RPDs, and the only opportunity for cooling is through convection by contact with the outside node enclosure, while other RPDs are packaged in shelves, where the number of RPDs is much larger, even requiring the need to support line-cards, and it is possible to implement cooling through forced air movement.

Therefore, while it is possible to maintain the HE equipment constant, the variety of MSO's HFC networks and application use cases, plus the desire to maintain a multi-supplier RPD ecosystem, drives the need for a large number of RPD types (e.g. upstream/downstream port counts and frequency splits), models and suppliers.

## 5. Solutions Attempted To Date

Over the last few years, as the development of RPDs proceeded through more use cases and newer models, suppliers have made every effort to reduce the number of implementations, and to the extent possible maintain a single RPD design. While the RPD hardware implementation is different between RPDs intended for different use cases, it is especially beneficial to maintain the same software base. To that end, most suppliers have tried to implement a single software base for their RPDs.

This is possible while the key hardware components used in the implementation of RPDs are the same or are of the same hardware generation. However, when the hardware implementation is different, it is no longer possible to maintain the same software base. Such is the case for RPDs that were implemented initially with discrete hardware components, and eventually were migrated to functionality integrated transceiver-type hardware devices, it is especially the case when migrating to generations of RPD hardware components that implement newer versions of the DOCSIS specifications, as is the case now with the advent of DOCSIS 4. In such cases, a change in the software base becomes inevitable.

To mitigate the need for software diversity, an approach was launched within the industry, led by CableLabs® and supported by a multitude of suppliers and operators, known as OpenRPD. The approach, implemented as an open-source project, consisted of developing an application layer software, which interfaced with the RPD hardware via a hardware abstraction layer and device drivers, and operated above a lower-level kernel software.

While all the above efforts were intended to maintain and/or arrive to a single software base, various ecosystem situations and changes in the hardware approach made it difficult to succeed.

## 6. Collaborating on a Single Software Solution

Given our industrial experience throughout the evolution described above, it became clear that many of the issues stemmed from the differences in the software implementation of the standard. The variability of behavior increases the complexity of managing and debugging the devices. So how could this be mitigated? The answer is, once again, to move to a single RPD software base. To accomplish this, a common RPD application software program was created to include all suppliers developing RPDs for the operator.

The idea of RPD suppliers sharing a common software base is not new. As described above, CableLabs hosted a similar program known as OpenRPD. The OpenRPD program allowed participating suppliers to access a common software base as well as submit changes for features and fixes. These suppliers showed early successes during the Remote PHY interoperability activities at CableLabs. Issues were certainly found during these activities, as expected. But when solutions to these issues were found, all participants reaped the benefits.

## 6.1. Chosing the Target Hardware

Before beginning the architectural design of the software, it is important to decide on the initial hardware that will be supported. This decision will have a large impact on the software development scope of work.

The primary question here is, should the currently deployed devices be considered in the software implementation? Or should the focus of the software be on only on new hardware development? Targeting both existing and new hardware would certainly reduce the mix of implementations to support, which is the goal of the project. But at the same time, it would greatly increase the scope of work necessary to deploy the software.

On the other hand, supporting only new hardware development could simplify the software architecture by leveraging the design of the new generation of integrated silicon solutions. Additionally, the new devices would soon be replacing the previous generation in the field. And because of these reasons, it was decided that focusing on the next generation of RPDs was the way to go.

## 6.2. RPD Software Architecture

Proper architecture of the software is critical when supporting multiple hardware targets. It is important to separate the main components are shared and those that will differ between the various platforms. Figure 3 below shows the basic architecture of the RPD software:

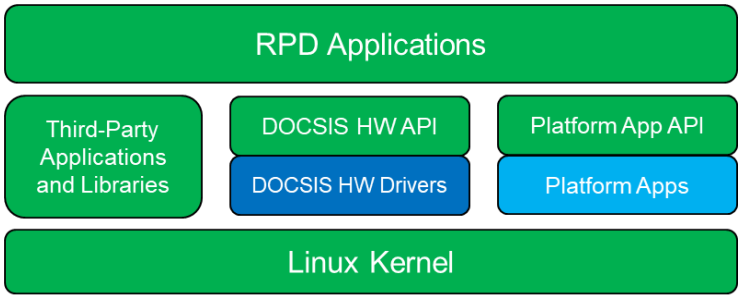


**Figure 3 – RPD Software Architecture**

The above blocks highlighted in green represent the functional components that are shared across all target platforms.

- **RPD Applications**: This is the collection of applications that provide the primary control interface and orchestrate control operations such as device configuration and monitoring.
- **DOCSIS HW API**: This specifies the programming interface for the DOCSIS-specific hardware. This includes operations such as DOCSIS channel and data-plane configuration.
- **Platform API**: This specifies the programming interface for the platform-specific hardware. This includes operations such as amplifier and attenuation configuration, LED control, timing configuration and monitoring, etc.
- **Third-Party Applications and Libraries**: This includes dependencies for features such as DHCP, SSH, 802.1x, etc.
- **Linux Kernel**: The distribution provides the supported Linux kernel version.

The blocks highlighted in shades of blue are specific to the hardware on the target platform.

- **DOCSIS Drivers**: These drivers conform to the interface defined by the DOCSIS API. They perform the low-level operations necessary to program the DOCSIS-specific hardware. Platforms that use the same DOCSIS hardware solution share the common drivers from the hardware provider.
- **Platform Applications**: These applications conform to the interface defined by the Platform API. These applications are specific to the each of the suppliers' hardware designs.

# 7. Managing the Software

The common RPD software will be used by multiple RPD suppliers targeting multiple platforms. This introduces some complexity when considering how the software base will be managed. Some of the questions raised are:

- Which build framework should be used to create the OS distribution?
- How should the software repositories be organized?
- How can each supplier separate their IP from the common software?

## 7.1. Deciding on the Build Framework

There are many choices when it comes to managing the project's build framework and operating system distribution. Some of the more popular frameworks for embedded systems are:

- Buildroot
- Yocto
- OpenWRT
- "Roll your own"

As expected, each of the choices come with its own advantages and disadvantages, briefly summarized below.

Buildroot is known for its simplicity. This means developers new to the framework can get started relatively quickly. However, precisely because of its simplicity, a significant amount of customization may be required to support many platforms within a single project.

The Yocto project, on the other hand, was designed with flexibility in mind. Yocto comes ready with a vast library supporting a large number of platforms. But with this flexibility comes complexity. Yocto is known for its steep learning curve.

The OpenWRT Project's primary focus is building firmware for commercial devices such as routers. It can be used for other types of embedded devices as well, but stepping away from its design for routers means additional customization.

Creating a custom distribution without a third-party framework is another option. This is something to consider if a project has very specific requirements that existing solutions do not provide. But this can introduce more overhead in maintenance when it comes to tasks such as keeping security patches up to date and resolving dependencies for upgrades.

Considering the choices above, the Yocto project was selected for managing the RPD program's OS distribution. In the end, it was decided that the time spent on learning the tools would be well worth the flexibility and hardware support required by the program.

## 7.2. The Yocto Project: Layers and Recipes

So, what makes the Yocto project flexible?

Let's start with what Yocto refers to as "recipes." A recipe is basically a script that provides instructions on how to build a particular entity (e.g. application, library, file system image, etc.). In its simplest form, a recipe for an application can just provide the location of the source software. Yocto can determine how to build the application and install it as long as the source is using one of the common automated build systems (e.g. autotools, cmake, meson, etc.).

Recipes are located in directory trees where the top directory is referred to as a "layer." A layer is a collection of recipes and configuration files. A single layer typically encompasses a particular category of recipes. For example, one layer might include recipes for shells, or even common RPD software applications. A typical project will have multiple layers, but will not necessarily use every recipe in every layer.

## 7.3. Organization of Layers in the RPD Project

The RPD project contains four categories of layers:

- **Third-Party Applications Layer**: This is actually several layers. The RPD project pulls in the existing layers from Yocto that provide the third-party open-source applications and libraries such as DHCP, SSH, and 802.1x. These layers are shared among all RPD projects.
- **Common RPD Applications Layer**: This layer contains the recipes for all of the common configuration and monitoring applications for the RPD. In addition, recipes for the DOCSIS HW and Platform APIs are provided. This layer is shared among all RPD projects.
- **SoC Applications Layer**: These layers contain the recipes for the various system on a chip (SoC) solutions available. They provide recipes for the required OS kernel and the DOCSIS HW drivers. Each RPD project selects the SoC layer associated with the hardware chosen for the target platform. Access control is necessary to restrict its use to suppliers that have the associated license agreements with the SoC provider.
- **OEM Applications Layer:** These layers contain recipes for applications, libraries, and drivers specific to the target device. Access control ensures that this layer is restricted to the associated supplier.

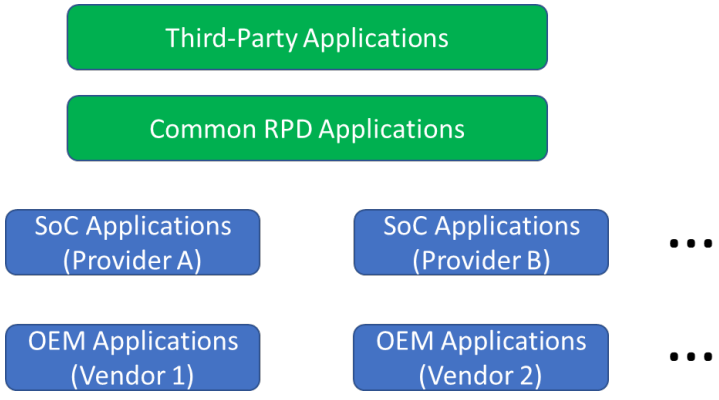


**Figure 4 – Yocto Layers for the RPD Project**

As described above, some of the layers are shared while others are selected based on the needs of the target device. For example, two different suppliers could use two different SoC solutions in their respective RPDs. Figure 5 below illustrates the combinations of layers required for an RPD developed by supplier "1" using SoC "B":

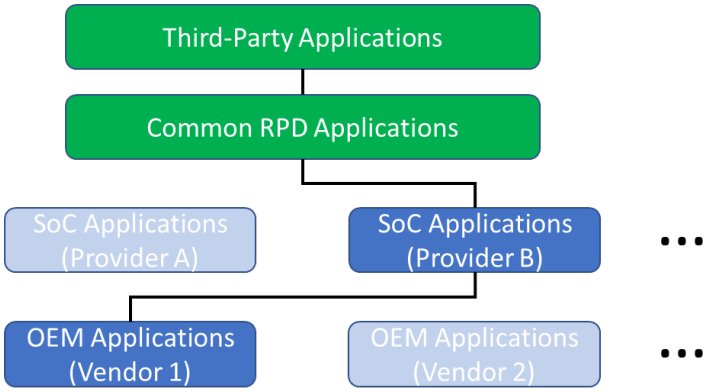


**Figure 5 – Example Selection of Yocto Layers**

### 7.4. Managing the Layers

As explained in the previous section, the Yocto layers are useful for organizing the build recipes into logical categories. These categories make sense from a software architectural standpoint. But they are also helpful when it comes to managing software changes and access control.

The layers in the RPD project have different access requirements depending on which category they fall into. For example, the common RPD applications must be accessible to all developers, but a specific supplier's OEM layer must not be available to other suppliers. To handle this requirement, each layer is placed in a separate repository. The proper access permissions can now be set uniquely for each repository.

One concern with this setup may be that developers now have to deal with a large number of repositories just for a single target. It is certainly true that the list of required layers can grow, especially given that just the third-party applications alone can be spread over many layers. But this is easily handled by using an application such as Google's "repo" command. Tools like this can take a single manifest that points to the collection of repositories for a project and manage any bulk operations such as checking out the software, branching, committing, etc. Operations on individual repositories can be performed as usual.

With the access controls in place for each of the repositories, this naturally decides which developers can see and make software changes as well as participate in software reviews for each layer. For example, only the suppliers with licensing agreements for a specific SoC can participate in the development for the associated layer. Likewise, all parties can participate in the development of the common RPD software layer.

## 8. What Comes Next?

Features continue to be added to the Remote PHY specifications that will need to be supported. The next big feature on the roadmap is streaming telemetry. This is a paradigm change from the pull model

supported today. Adding the vast number of statistics required for monitoring field deployments is no small task. But the results of this effort can be shared among those involved with the project.

Another task on the horizon is virtualization of the RPD control plane. Running the RPD software without the underlying hardware can have multiple uses. A virtual version of the RPD can be used for automated integration testing in the build environment. Additionally, initial development testing for some features can be started with the virtual RPD, reducing the time spent on lab resources.

Overall, the roadmap may be similar to other RPD software programs. The big benefit with this program is that the fruits are shared by all.

## 9. Conclusion

Managing a large deployment of Remote PHY devices is a challenging task. Problems will arise in any large network. Troubleshooting these issues become more complex when the devices differ in behavior, debug capabilities, and supported features. Moving to a common software base certainly will not solve all issues typically seen in deployments, but standardizing the implementation of the management interface will simplify the configuration, monitoring and troubleshooting of RPDs by reducing the supplier-specific behavior. Accomplishing this goal is possible with a software architecture and development environment that supports multi-supplier collaboration.

# Abbreviations

| | |
|---|---|
| RPD | remote PHY device |
| RPHY | remote PHY |
| DOCSIS | data-over-cable service interface specifications |
| API | application programming interface |
| PHY | physical layer |
| SoC | system on a chip |