UNLEASH THE POWER OF LIMITLESS CONNECTIVITY
Wireline Access Network

A Common Remote PHY Software Stack for all RPDs?

Michael Robinson
Remote PHY Software Architect
Comcast Cable Communications
Agenda

• Software Fragmentation and DAA
• Mitigating the Problem
• RPD Software Architecture
• Build Framework
• The Yocto Project
• What Comes Next?
• Summary
What is Software Fragmentation?

Consider the example of a web application:

- Design to web standards (HTML, CSS, PNG)
- In what environments will this application be deployed?
- What web browsers will be supported?
- What hardware platforms will be supported?
- What operating systems will be supported?
- What versions will be supported in each of these environments?
- The number of test scenarios grows quickly
CMTS-RPD Interoperability Scenarios

- In the simplest case, a CMTS Core is connected to RPDs with identical hardware and software (no fragmentation)
- Next consider this same CMTS Core model connected to a set of RPDs from a different supplier
- Now consider a CMTS Core from a different supplier in similar deployments
- Like the web application example, the number of test scenarios grows quickly
Mitigating the RPD Software Fragmentation Issue

Moving to a Single Software Solution

- Reducing the diverse software implementations across RPDs has several advantages
  - Model-specific hooks in the Core’s code can be reduced
  - Troubleshooting can be less complex when there is less model-specific behavior
  - The number tests for model-specific behavior can be reduced

- The answer is to move to a single RPD software base
  - A common RPD code base is shared by multiple suppliers
  - A collaborative development effort by the MSO and suppliers
  - Similar to the CableLabs® OpenRPD project except managed by the MSO
  - Reduces the duplication of effort among suppliers while aligning behavior
RPD Applications include:
- System startup orchestration
- Primary control interface (GCP and L2TPv3)
- Device configuration (e.g. port/channel configuration)
- Device monitoring (e.g. statistics, status, alarms)
- Software upgrades
- Device reboot
Third-Party Applications include common open-source software for features such as:

- 802.1x
- DHCP
- SSH
- TFTP
- HTTP
DOCSIS Hardware Drivers and API

DOCSIS HW API:
• Defines a common interface for all DOCSIS hardware solutions
• Common for all platforms

DOCSIS HW Drivers:
• Implement the DOCSIS HW API functionality
• Specific to the underlying hardware
• Shared among platforms that use the same DOCSIS hardware
Platform Application API:
• Defines a common interface for functionality requiring platform-specific implementation
• For example, RF power/attenuation configuration, LED control, etc.

Platform Applications:
• Implementation of the Platform Application API functionality
• Specific to the manufacturer
Linux Kernel:
- Common Linux kernel version
- Kernel configuration is customized as necessary
Deciding on the Build Framework

• The build framework is used to create the project’s distribution, including the kernel and the content of root file system

• The following frameworks were considered:
  • **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.
The Yocto Project

The Yocto Layer Model

Yocto’s Layer Model lends itself well to a multi-vendor collaborative software project:

“The Layer Model is designed to support both collaboration and customization at the same time. Layers are repositories containing related sets of instructions which tell the build system what to do. Users can collaborate, share, and reuse layers.”

-The Yocto Project
Yocto Recipes and Layers

**Recipes**
- Provide the instructions for building a package (e.g. a single application)
- In its simplest form, a recipe can simply point to the location of the code source
- In many cases, Yocto will automatically determine how to build and install the recipe’s package

**Layers**
- Each layer contains a collection of recipes
- Layers are maintained in a separate repositories
- A recipe located in one layer can be customized in another layer
Layers in the RPD software project align with the code architecture:

- **Third-Party Applications**
  - Yocto provides several layers of open-source applications

- **Common RPD Applications**
  - Single layer containing the recipes for the RPD functionality for all RPD platforms

- **SoC Applications**
  - One layer for each DOCSIS hardware provider

- **OEM Applications**
  - One layer for each supplier’s device type
  - A supplier can optionally combine devices in a single layer
Example Selection of RPD Yocto Layers

The example shown illustrates the layer selection for an RPD project by supplier 1 using a SoC from provider B.

Since each of the layers are maintained in a separate repository, maintaining proper access rights helps manage what code can be seen by each supplier.

In the example shown, supplier 1 cannot access supplier’s 2 OEM application content. Additionally, if supplier 1 does not have a license for SoC provider A’s product, then the associated layer will be inaccessible.
What Comes Next?

On the Roadmap

• Streaming Telemetry
  • This is a paradigm change from the pull model supported today
  • The gNMI protocol will be used to push statistics to remote collectors
• Virtualization of the RPD’s control plane
  • Run the RPD software without the underlying hardware
  • Can be used for integration testing and as a development platform
Summary

In Conclusion

The common software:

- Moves toward a more consistent behavior for deployed RPDs
- Reduces model-specific customizations in the Core
- Reduces model-specific test variations
- Encourages collaboration among vendors, reducing duplicated effort
Thank You!

Michael Robinson
Remote PHY Software Architect
Comcast Cable Communications
michael_robinson2@comcast.com