How to Choose a Software Update Mechanism for Embedded Linux Devices

Leon Anavi
Konsulko Group
leon.anavi@konsulko.com
leon@anavi.org
Embedded Linux Conference North America 2022
Konsulko Group

- Services company specializing in Embedded Linux and Open Source Software
- Hardware/software build, design, development, and training services
- Based in San Jose, CA with an engineering presence worldwide
- http://konsulko.com/
Agenda

- Things to consider
- Common embedded Linux update strategies
- Open source solutions
- Practical examples with Yocto/OpenEmbedded
- Conclusions
Things to Consider: Update Size

- Are there any limitations of the disk space?
- Are there any limitations of the network bandwidth for the data transfer?

- According to cable.co.uk in the US the average cost of mobile data is $3.33 per 1GB

https://www.cable.co.uk/mobiles/worldwide-data-pricing/
Things to Consider: Data Transfer

What are the different ways of transferring data for the updates?

- Over the air using WiFi or cellular data
- Ethernet cable
- USB stick
- Something else
Things to Consider: Managing Devices

- Do you need to update a single device or multiple devices?
- Are all of devices using the same software stack?
- Are all devices online and do you need to monitor the update of the devices?
- Do you need to update different devices at different times?
- How to update fleet of devices?
Things to Consider: Building Images

- What distribution and build system do you use?
- Is there a BSP for the hardware you use?
- Is the software update technology compatible with the build system and the BSP?
Popular open source build systems for custom embedded Linux distributions:

- Yocto/OpenEmbedded
- Buildroot
- PTXdist
- OpenWRT
- Other
Can I just use Debian?

- Debian is a stable full distribution with tens of thousands of packages available as binary files for installation without the need to cross-compile from source
- Numerous Debian derivatives exist for embedded devices

Debian or Yocto Project? Which is the Best for your Embedded Linux Project?
Chris Simmonds, Embedded Linux Conference Europe 2019
https://www.youtube.com/watch?v=iDlIXa8SzUgr
The Yocto Project

- Open source collaborative project of the Linux foundation for creating custom Linux-based systems for embedded devices using the OpenEmbedded Build System
- OpenEmbedded Build System includes BitBake and OpenEmbedded Core
- Poky is a reference distribution of the Yocto Project provided as metadata, without binary files, to bootstrap your own distribution for embedded devices
- Bi-annual release cycle
- Long term support (LTS) release covering two-year period
Common Embedded Linux Update Strategies

- A/B updates (dual redundant scheme)
- Delta updates
- Container-based updates
- Combined strategies
A/B Upgrades

- Dual A/B identical rootfs partitions
- Data partition for storing any persistent data which is left unchanged during the update process
- Typically a client application runs on the embedded device and periodically connects to a server to check for updates
- If a new software update is available, the client downloads and installs it on the other partition
- Fallback in case of update failure
Delta Updates

- Only the binary delta between the difference is sent to the embedded device
- Works in a Git-like model for filesystem trees
- Saves storage space and connection bandwidth
- Rollback of the system to a previous state
## A/B vs Delta Updates

<table>
<thead>
<tr>
<th>Update Strategy</th>
<th>Storage Space</th>
<th>Update Size</th>
<th>Rollback to a Previous Stage</th>
<th>Fallback to a Back-up Image on separate partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/B Updates</td>
<td>Large</td>
<td>Large</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Delta Updates</td>
<td>Small</td>
<td>Small</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Container-based Updates

- Container technology has changed the way application developers interact with the cloud and some of the good practices are nowadays applied to the development workflow for embedded Linux devices and Internet of Things.
- Containers make applications faster to deploy, easier to update and more secure through isolation.
- Yocto/OE layer meta-virtualization provides support for building Xen, KVM, Libvirt, docker and associated packages necessary for constructing OE-based virtualized solutions.
Combined Strategies

Multiple combinations exist for more complicated use cases, for example:

- A/B updates with delta updates
- Containers with A/B updates of the base custom embedded Linux distribution
Popular open source solution for updates

- Mender
- RAUC
- SWUpdate
- Swupd
- UpdateHub
- Balena
- Snap
- libostree (OSTree)
- Aktualizr
- Aktualizr-lite
- QtOTA
- Torizon
- FullMetalUpdate
- Rpm-ostree (used in Project Atomic)
Mender

- Available as a free open source or paid commercial/enterprise plans
- A/B update scheme for open source and all plans as well as delta updates for professional and enterprise plans
- Back-end services (Hosted Mender)
- Written in Go, Python, JavaScript
- Yocto/OE integration through meta-mender and extra BSP layers: https://github.com/mendersoftware/meta-mender
- Source code in GitHub under Apache 2.0
The following hardware platforms and development boards are supported:

- Raspberry Pi
- BeagleBone
- Intel x86-64
- Rockchip
- Allwinner
- NXP
- And more in: https://github.com/mendersoftware/meta-mender-community
meta-mender-community
RAUC

- A lightweight update client that runs on an Embedded Linux device and reliably controls the procedure of updating the device with a new firmware revision
- Supports multiple update scenarios
- Provides tool for the build system to create, inspect and modify update bundles
- Uses X.509 cryptography to sign update bundles
- Compatible with the Yocto Project, PTXdist and Buildroot
RAUC Licenses

- RAUC – LGPLv2.1
  https://github.com/rauc/rauc
- meta-rauc - MIT
  https://github.com/rauc/meta-rauc
- rauc-hawkbit – LGPLv2.1
  https://github.com/rauc/rauc-hawkbit
- rauc-hawkbit-updater – LGPLv2.1
  https://github.com/rauc/rauc-hawkbit-updater
RAUC Integration Steps

- Select an appropriate bootloader
- Enable **SquashFS** in the Linux kernel configurations
- **ext4** root file system (RAUC does not have an ext2 / ext3 file type)
- Create specific partitions that matches the RAUC slots
- Configure Bootloader environment and create a script to switch RAUC slots
- Create a certificate and a keyring to RAUC’s **system.conf**
- Supports single and redundant data partitions
- For redundant data partitions the active rootfs slot has to mount the correct data partition dynamically, for example with a udev rule
meta-rauc-community

- Yocto/OE layer with examples how to integrate RAUC on various machines
- Started in 2020
- Moved to the RAUC organization in GitHub in 2021
- Currently supports x86-64, QEMU, Raspberry Pi through `meta-raspberrypi`, Sunxi (Allwinner) devices through `meta-sunxi`, NVIDIA Jetson TX2 through `meta-tegra`
- https://github.com/rauc/meta-rauc-community/

**Contributions are always welcome as GitHub pull requests!**
Integration layer (branch **Dunfell**):

Add layers to **bblayers.conf** and add the following configuration to **local.conf**:

```bash
MACHINE = "raspberrypi4"
DISTRO_FEATURES_append = " systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"
VIRTUAL-RUNTIME_initscripts = ""
IMAGE_INSTALL_append = "rauc"
IMAGE_FSTYPES="tar.bz2 ext4 wic.bz2 wic.bmap"
SDIMG_ROOTFS_TYPE="ext4"
ENABLE_UART = "1"
RPI_USE_U_BOOT = "1"
PREFERRED_PROVIDER_virtual/bootloader = "u-boot"
WKS_FILE = "sdimage-dual-raspberrypi.wks.in"
```
Manual RAUC Update of Raspberry Pi 4

- On the build system:
  
  ```
  cd tmp/deploy/images/raspberrypi4/
  python3 -m http.server
  ```

- On the embedded device:
  
  ```
  rauc install /tmp/update-bundle-raspberrypi4.raucb
  reboot
  ```
Eclipse hawkBit

- Domain independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP based networking infrastructure
- Written in Java
- Available in GitHub under EPL-1.0 License
- Compatible with RAUC and SWUpdate
- https://www.eclipse.org/hawkbit/
Mender

Mender A/B updates supports two client modes:

- Managed (default) - client running as a daemon polls the server for updates
- Standalone - updates are triggered locally which is suitable for physical media or any network update in pull mode

```shell
SYSTEMD_AUTO_ENABLE_pn-mender = "disable"
$ cd tmp/deploy/images/raspberrypi4
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
$ mender -install http://example.com:8000/core-image-base-raspberrypi4.mender
```
Mender Data Partition

- Mender creates a `/data` partition to store persistent data, preserved during Mender updates. Supports ext4, Btrfs and F2FS file systems.

- The Mender client on the embedded devices uses `/data/mender` to preserve data and state across updates.

- Variable `MENDER_DATA_PART_SIZE_MB` configures the size of the `/data` partition. By default it is 128 MB. If enabled, mender feature `mender-growfs-data` which relies on `systemd-growfs` tries to resize on first boot with the remaining free space.

- It is possible to create an image for the data partition in advance with bitbake:

  ```
  IMAGE_FSTYPES:append = " dataimg"
  ```
Steps to install Mender A/B update on embedded Device:

- Apply update
- Reboot
- On the first boot after a successful update, though the Mender client a commit must be performed to accept the update (otherwise the system will roll-back on next reboot)
Deployment of a single file, directory or even a container image is possible through “Application updates”
Mender supports several add-ons:

- **Remote Terminal** - interactive shell sessions with full terminal emulation
- **File Transfer** - upload and download files to and from a device
- **Port Forward** - forward any local port to a port on a device without opening ports on the device
- **Configure** - apply configuration to your devices through a uniform interface
Mender with x86-64 support

- Mender added support for x86-64 machines through GRUB in 2018
- Initial installation of the distribution is most commonly done using a live image on a USB stick
Mender Delta Updates

- Mender offers robust delta update rootfs as a module for the commercial Mender plan (closed source implementation)
- Requires reboot to apply the update
- Supports rollback
- Tool mender-binary-delta create a binary delta by comparing 2 different Mender artifacts
- Mandatory requirement for the implementation is a read-only root file system
Yocto and OpenEmbedded offer two options to create a read-only root filesystem:

- Thought the image’s recipe file:

  ```
  IMAGE_FEATURES += "read-only-rootfs"
  ```

- Alternatively, through `local.conf`:

  ```
  EXTRA_IMAGE_FEATURES = "read-only-rootfs"
  ```

- Beware, there might be packages in the image that expect the root filesystem to be writable and might not function properly. A solution is to move these files and directories to the data partition.
Combined Strategies with Containers

- Yocto/OE layer **meta-virtualization** provides support for building Xen, KVM, Libvirt, docker and associated packages necessary for constructing OE-based virtualized solutions

- **virtualization** has to be added to the **DISTRO_FEATURES**:

  ```
  DISTRO_FEATURES:append = " virtualization"
  ```

- For example adding Docker to the embedded Linux distribution is easy:

  ```
  IMAGE_INSTALL:append = " docker-ce"
  ```

- There are use cases on powerful embedded devices where containers are combined with A/B updates of the base Linux distribution built with Yocto/OE, for example with RAUC or Mender
Libostree (previously known as “OSTree”), is a shared library and tools for distributing and deploying file system images as atomic upgrades of the whole file system tree.

- Provides a suite of command line tools that combines a “git-like” model for committing and downloading bootable filesystem trees, along with a layer for deploying them and managing the bootloader configuration.
- Effective with respect to disk space and connection bandwidth.
- Supports rollback of the system to a previous state.
- Used in many embedded Linux solutions: Fedora IoT, Torizon, Aktualizr, Aktualizr-lite and QtOTA.
libostree

- libostree encourages systems to implement UsrMove
  [https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/](https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/)

- libostree comes with optional dracut+systemd integration:
  [https://ostreedev.github.io/ostree/adapting-existing/](https://ostreedev.github.io/ostree/adapting-existing/)

- libostree only preserves /var across upgrades

- /var is empty by default and the operating system needs to dynamically create the targets of these at boot, for example with systemd-tmpfiles (if using systemd)
Aktualizr and Aktualizr-lite with libostree

- Aktualizr is a C++ application that implements the client-side functionality for OTA Connect according to the Uptane security framework.

- libaktualizr provides API calls to perform the various steps that are necessary for checking for updates, validating the files downloaded, etc.
  https://github.com/advancedtelematic/aktualizr

- Aktualizr-lite is a C++ implementation of TUF OTA update client based on aktualizr without the complexity of Uptane.
  https://github.com/foundriesio/aktualizr-lite

- Yocto/OE integration:
  https://github.com/advancedtelematic/meta-updater
  https://github.com/foundriesio/meta-lmp
Conclusion

- There are numerous things to consider when implementing an upgrade mechanism for an embedded Linux device.
- There are many reliable open source software solutions to upgrade embedded Linux devices to use and worth developing another proprietary homegrown solution.
- Combined update strategies such as A/B upgrades with containers for applications are increasingly popular nowadays.
- The update process implementation depends on the build system and the bootloader.
- Often real-world solutions require a persistent data partition which is left unchanged during the update process.
Thank You!

Useful links:

- https://www.yoctoproject.org/
- https://mender.io/
- https://rauc.io/
- https://ostreedev.github.io/ostree/
- https://www.konsulko.com/how-mender-works/