Migrating to Yocto: A Guide and Lessons Learned

Muhammad Tauqir Ahmad
About Cisco Meraki
Our Products

- MR Wireless access points
- MS Switches
- MX Security appliance router
- MV Security cameras
Our Products and Variations

CPU Architectures

- Intel
- ARM
- PowerPC

SoC Vendors

- NXP
- Broadcom
- Qualcomm

Other chip vendors

- Gemalto
- Marvell

Linux kernels

- 4.4: blurry fish butt
- 3.14: shuffling zombie juror
- 3.4: saber-toothed squirrel
- 4.9: roaring lionus
Repository Layout

Single local repository for everything

- Main entry point for builds
- Product-specific directories
  - All Linux versions
- User-space apps
- OpenWRT checkout
- Vendor SDKs
- Wireless drivers
- Packet processing engine
- Third-party software
- More drivers

router
  ├── base
  │   ├── Makefile
  │   │   ├── mx60
  │   │       └── mv12
  │   └── linux-3.14
  │       └── apps
  │           ├── openwrt
  │           └── vendors
  │                   └── wlan-10.4
  └── apps
      └── extern
          └── quickassist
Legacy Build System

- Built around OpenWRT
- Single Makefile to rule them all
  - Hand-written Makefile
  - 13 years of cruft
- Typical build
  
  $ make -j16 BOARD=mx60
Why Migrate?
Unmaintainable OpenWRT

OpenWRT @ 2006

Our OpenWRT @ 2019

OpenWRT @ 2019
Legacy Builds are Painful

- Builds are slow
  - A typical clean build takes ~4 hours
  - As a result of undeclared dependencies, builds are linear
- Git pull often breaks builds
  - Have to run `git clean -xfd` followed by a full rebuild
  - When in doubt, `git clean`
New Product

- We were starting to bring-up a new product
- We get thorough testing for free
- Good vendor support for Yocto
Summary

- Old build system
  - Unmaintainable legacy build system
  - Slow, linear builds

- Need a new build system
  - Fast
  - Flexible enough to support our use cases

- So we decided to give Yocto a shot!
  - Modern, fast; based on BitBake
  - Better equipped to handle complex use cases
  - Good vendor support
First (Bit)Bake
Follow the Quick Start Guide

Yocto Project Quick Start

$ bitbake core-image-minimal

$ runqemu qemux86
Build Your Own Image

1. Add your new **layer**, sift in a new **distro**, add your **machine**
2. Combine with a **package group** with a few essentials
   
   ```
   RDEPENDS = "busybox curl dropbear dhcpd"
   ```
3. Mix with your own new **image**
   
   ```
   inherit core-image
   IMAGE_INSTALL = "packagegroup-meraki"
   ```
4. Get baking!
   
   ```
   MACHINE=mx67 bitbake meraki-image
   ```

But it’s missing Meraki-specific stuff!
Port Packages from OpenWRT to Yocto
Porting from OpenWRT

- Some packages missing in Yocto
- Others built different in Yocto and OpenWRT
- Replicate the build in Yocto ...
- ... to minimize differences during transition from OpenWRT to Yocto

But first, a BitBake crash course!
Refresher: Recipes

- Contains a **set of instructions** to build something
- Parsed and executed by bitbake to produce a **package**
- Takes the form `curl_1.4.31.bb`

\[ PV = "1.4.31" \]
Refresher: Appends

- Modifies/extends a **recipe**
- Can apply to a specific version or wildcard
  - `curl_1.4.31.bbappend`
  - `curl_% bbappend`

![Diagram showing the process of appending to a recipe and building a package](image-url)
Refresher: Layers

- Segregate logical sets of recipes & appends
- You can pick and choose layers

```
BBLAYERS += "poky/meta/ \n             meraki/meta/"
```

```
poky/meta/
  Recipe
  curl.bb
  Append
  curl_%bbappend
meraki/meta/
```

```
+ bitbake curl
```

```
Package
curl.deb
```
Back to Porting Packages

- Match versions
- Apply the same patches
- Use the same configure options
- Use the same compiler flags
Add an Append

curl_1.4.31.bb:
...
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"

poky/meta/

curl_%bbappend:

meraki/meta/
Match Versions

curl_1.4.31.bb:
...
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"

poky/meta/

curl_%_bbappend:
1 PV = "1.4.39"

meraki/meta/
Match Versions

curl_1.4.31.bb:
...
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"

poky/meta/

curl_%bbappend:
1 PV = "1.4.39"
2 LIC_FILES_CHKSUM = "..."
3 SRC_URI[md5sum] = "..."
4 SRC_URI[sha256sum] = "..."

meraki/meta/
Apply the Same Patches

curl_1.4.31.bb:
...
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"
poky/meta/

curl_%bbappend:
1 PV = "1.4.39"
2 LIC_FILES_CHKSUM = "..."
3 SRC_URI[md5sum] = "..."
4 SRC_URI[sha256sum] = "..."
5 SRC_URI += ".../meraki.patch"

meraki/meta/
Use the Same Configure Options

curl_1.4.31.bb:
...  
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"

poky/meta/

curl_%bbappend:
1 PV = "1.4.39"
2 LIC_FILES_CHKSUM = "...
3 SRC_URI[md5sum] = "...
4 SRC_URI[sha256sum] = "...
5 SRC_URI += ".../meraki.patch"
6 EXTRA_OECONF = "--enable-ipv6"

meraki/meta/
Use the Same Compiler Flags

curl_1.4.31.bb:
...
SRC_URI = ".../curl_${PV}.gz"
...
EXTRA_OECONF = "--enable-cares"
TARGET_CFLAGS += "-DFLAG=1"
poky/meta/

curl_%bbappend:
1 PV = "1.4.39"
2 LIC_FILES_CHKSUM = "..."
3 SRC_URI[md5sum] = "..."
4 SRC_URI[sha256sum] = "..."
5 SRC_URI += ".../meraki.patch"
6 EXTRA_OECONF = "--enable-ipv6"
7 TARGET_CFLAGS_remove = "-DFLAG=1"
8 TARGET_CFLAGS += "-DFLAG=0"
meraki/meta/
Porting from OpenWRT

- Seems fairly straightforward
- But some packages weren’t in Yocto
- And not every port was this simple
- And we had ~50 packages to port!
- And not enough time
Porting Day – 1 week

- Label all tasks easy, medium, hard
- Provide a warm `sstate-cache` to speed up builds
- Ask everyone to do a build before the day
- Send out invites
Give a Tutorial
# Use Task Management Tools

**Product**: PROD-438 Port packages from OpenWRT to Yocto / PROD-471

## Port nettle

<table>
<thead>
<tr>
<th>Details</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong>: Sub-task</td>
<td>Assignee: Unassigned</td>
</tr>
<tr>
<td><strong>Status</strong>: TO DO (View Workflow)</td>
<td>Assign to me</td>
</tr>
<tr>
<td><strong>Priority</strong>: Medium</td>
<td><strong>Reporter</strong>: Muhammad Tauqir Ahmad</td>
</tr>
<tr>
<td><strong>Resolution</strong>: Unresolved</td>
<td><strong>Component/s</strong>: None</td>
</tr>
</tbody>
</table>
Use Chat

15:26 Vincent Huang 😊 there is a patch in a directory called `readline-7.0` in poky that needs to be applied, but we build `readline-6.3` so bitbake can’t find the patch file resolution: “just copy the patch over” - tauqir CR for readline please? https://gerrit.ikarem.io/c/50505/

14:20 Paul Crandell quick CR plz https://gerrit.ikarem.io/c/50480/
14:32 Younghwan Go CR for bc? https://gerrit.ikarem.io/c/50485/ I only had to change SRC_URI
14:46 Paul Crandell FILES_${PN} += "${bindir}" this seems to be pulling files from `router/base/somefolder/` What should bindir be instead?
15:19 Vincent Huang 😊 is there an equivalent of `$(THISDIR)` for referring to the `$(THISDIR)` of the `.bb` file that you’re appending to? (in a `.bbappend`)
15:26 Tauqir Ahmad @vincentth What do you need it for? Maybe I have an alternate solution.
Great Success!

- 35 recipes ported!
- Many of the medium and hard ones too
- Huge help with getting **people on board**
Tip: Recruit People

- Helps reduce the friction from change
- Meet your own objectives quicker
Extend Yocto for Your Needs
Extend Yocto for Your Needs

- Yocto might not have everything for your specific use-cases
- But you can use Yocto-provided primitives to build new tools ...
- ... to help with your specific use-cases

Example Meraki use-cases

- Building from disk
- Enable/disable features per-product
Building from Disk: EXTERNALSRC

meraki-apps.bb:

inherit externalsrc

EXTERNALSRC = "/path/to/sources"

- Packages create files in the source directory
- Multiple sources
- Complete rebuild if sources change
Building from Disk: SRC_URI

meraki-apps.bb:

SRC_URI = "file://path/to/source1 file://path/to/source2"

S = "${WORKDIR}/path/to/sources"

- Multiple sources
- Setting S is annoying
- Complete rebuild if sources change
We Built Merakisrc

meraki-apps.bb:

```
inherit merakisrc
MERAKISRCS = "/path/to/source1 /path/to/source2"
do_configure[patterns] = "*.in *.ac *.am CMakeLists.txt *.cmake"
```

✅ Multiple sources

✅ Copies sources
   • Through prefuns, instead of do_fetch

✅ Step only re-run if relevant files change
   • Through file-checksums, handled by sstate
Machine & Distro Features

- Yocto provides `MACHINE_FEATURES & DISTRO_FEATURES`
- But they didn’t map to features we want to enable/disable

Diagram:
- **Distro Features**
  - ppp
  - alsa
  - x11
  - bluetooth
  - ...

- **Meraki Features**
  - bird
  - webserver
  - LEDs
  - LTE
  - ...

Meraki
We Built Meraki Features

- We introduced our own variable: `MERAKI_FEATURES`
- Controls which packages get installed, configure options for recipes

```plaintext
mx67.conf:
MERAKI_FEATURES += "bird wireless vpn"

packagegroup-meraki.bb:
RDEPENDS_${PN} += " \n${@utils.contains("MERAKI_FEATURES", "bird", \n  "${BIRD_RDEPENDS}" , "")}"
Room for Improvement
Problem: Mismatched C Library Headers

Machine running Linux 3.14

$ ethtool -S eth0
The Error

The error message indicates that in the function `string generate_stats_resp(const string&)`, a case statement is trying to use a variable `SPEED_5000` which is not declared in the current scope.

```cpp
| meraki/poder.cc: In function 'string generate_stats_resp(const string&)':
| meraki/poder.cc:1004:30: error: 'SPEED_5000' was not declared in this scope
|    case SPEED_5000:  speed_5000m |= mask; break;
|    ^
| CMakesFiles/poder.dir/build.make:65: recipe for target 'poder.cc.o' failed

recipes-meraki-apps/poder/poder.bb:do_compile failed with exit code '1'
```
The User-space App

poder: add 5G bitfield

This commit adds the speed_5000m_bitfield in order to be able to report 5G uplink on the firmware side.

diff --git a/meraki/poder.cc b/meraki/poder.cc
@@ -1035,6 +1037,7 @@
  case SPEED_1000: speed_1000m |= mask; break;
  case SPEED_100:  speed_100m  |= mask; break;
  case SPEED_2500: speed_2500m |= mask; break;
+     case SPEED_5000: speed_5000m |= mask; break;
  }
  if (edata.duplex == DUPLEX_FULL) {
    duplex |= mask;

}
The Modified Kernel

linux-3.14: add 5G speed to ethtool.h

In future commits, I intend to check for 5G uplink in poder_agent.cc.

linux-4.4 is the first kernel that has SPEED_5000, so I am adding it to previous kernel versions.

diff --git a/linux-3.14/include/uapi/linux/ethtool.h b/linux-3.14/include/uapi/linux/ethtool.h
@@ -974,6 +974,7 @@
enum ethtool_sfeatures_retval_bits {
    #define SPEED_100              100
    #define SPEED_1000             1000
    #define SPEED_2500             2500
+#define SPEED_5000             5000
    #define SPEED_10000            10000
    #define SPEED_UNKNOWN          -1
Problem: Mismatched C Library Headers

Machine running Linux 3.14

novation/meta/
linux_4.1
linux-libc-headers_4.1
make headers_install
libc
poder

poky/meta/
linux_3.14

Meraki
Solution: Use the Same Sources

Machine running Linux 3.14

`$ poder`
Solution: Use the Same Sources

```
linux-libc-headers_4.1.bb
SRC_URI = ".../linux_${PV}.gz"
...
...
do_install() {
    oe_runmake headers_install
}
...
poky/meta/
```

```
linux-libc-headers.bb
PV = "${PREFERRED_VERSION_kernel}"
```
Solution: Use the Same Sources

```
linux-libc-headers_4.1.bb
SRC_URI = ".../linux_${PV}.gz"
...
...
do_install() {
    oe_runmake headers_install
}
...

poky/meta/
```

```
linux-libc-headers.bb
PV = "${PREFERRED_VERSION_kernel}"
S = "${STAGING_KERNEL_DIR}"

meraki/meta/
```
Solution: Use the Same Sources

```bash
linux-libc-headers_4.1.bb
SRC_URI = ".../linux_${PV}.gz"
...
do_install() {
  oe_runmake headers_install
}
...
poky/meta/
```

```bash
linux-libc-headers.bb
PV = "${PREFERRED_VERSION_kernel}"
S = "${STAGING_KERNEL_DIR}"
...
do_install() {
  oe_runmake headers_install
}
...
meraki/meta/
```
Lesson: Some Hacks are Okay

- The flexibility of Yocto means hacks are not too hard
- Some hacks are okay to make progress
- Work with the community towards a neater long-term solution
Room for Improving Layers
Layers Are Not Programmatically Selectable

```bash
$> MACHINE="mx67" bitbake meraki-image
```

```
build/conf/

bblayers.conf

BBLAYERS =

meta-mx67

meraki/meta/

meta-openembedded/meta-oe/

poky/meta-poky/

poky/meta/
```
Layers Are Not Programmatically Selectable

```bash
$> MACHINE="mr55" bitbake meraki-image
```

```
build/conf/

bblayers.conf

BBLAYERS =

meta-mr55/
meraki/meta/
meta-openembedded/meta-oe/
poky/meta-poky/
poky/meta/
```
But They are Still Useful for Backports
But They are Still Useful for Backports

curl_1.5.0

meraki/meta/

meta-oe/meta-oe/: 2.6

poky/meta-poky/: 2.6

poky/meta/: 2.6
Successes!
Successes: Build Times

- \(~4\text{ hours} \rightarrow ^{15}\text{ minutes}\) for a clean build!
Successes: Most Products Using Yocto

More than half our products are on Yocto!
Successes: Easy Maintenance

- Yocto is super easy to upgrade
- We don’t have to maintain our own recipes
- We don’t have to hunt patches
Closing Thoughts

- Yocto is very well designed & flexible
- We would love to open source our work
- Community could adopt a formal RFC process
- Get people on board
Thank you!

- Ram Subramanian
- Mat Prokos for the PoC!
- Everyone who helped me prepare the presentation
- The amazing Yocto community!
- All the folks who organized Yocto dev days 2018
- And all the folks who answered our questions there!

Email me at t@meraki.com
Questions?