Poky-tiny and Beyond, or Trying to put the Yocto in Yocto Project

Scott Murray
scott.murray@konsulko.com

Konsulko Group

About Me

- Linux user/developer since 1996
- Embedded Linux developer starting in 2000
- Principal Software Engineer at Konsulko Group
- 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
  - https://konsulko.com

Agenda

- Quick overview of OpenEmbedded / Yocto Project
- Review of embedded Linux distribution size history
- poky-tiny exploration
- Sizes of some common image features / packages
- Other image size reduction options
- Summary / Recommendations
- ...and Beyond?

Caveats

- Not going to dig into kernel size reduction techniques, as that topic receives frequent coverage
- Not covering sizes of graphical desktop components
- Test builds for size investigation were done with the qemux86 architecture

OpenEmbedded & The Yocto Project in 1 Slide

- OpenEmbedded (OE) is a build system and associated metadata to build embedded Linux distributions.
- The Yocto Project is a collaboration project that was founded in 2010 to aid in the creation of custom Linux based systems for embedded products. It is a collaboration of many hardware and software vendors, and uses OpenEmbedded as its core technology. A reference distribution called “poky” (pock-EE) built with OE is provided by the Yocto Project to serve as a starting point for embedded developers.

Yocto (symbol **y**) is a unit prefix in the metric system denoting a factor of 10^{-24} or 0.000000000000000000000001.

- *Yocto-* in Wikipedia, retrieved March 11, 2018 from https://en.wikipedia.org/wiki/Yocto-

A brief history of ~~time~~size

- When Linus Torvalds first released Linux in 1991, typical desktop PCs would have been Intel 386 or 486 based, running at 25 - 40 MHz, with 4 - 32 MB of RAM, and had a hard disk of at most a few hundred MB (if they had one at all).
- When embedded Linux started taking off in the late 90s and early 00s, the typical target platforms had faster CPUs (100s of MHz), but RAM and storage were still constrained by cost to sizes like 8 - 64MB RAM and 8 - 16MB flash memory.
- Embedded Linux distributions of that era tended to be built with custom in-house build systems, or using commercial offerings from companies like MontaVista, TimeSys, etc.
- Small embedded Linux distributions would typically be based on the kernel, uclibc, and BusyBox...

A brief history of ~~time~~size (2001-2002)

- In 2002, the Linux based Sharp Zaurus **SL-5500** was released outside Japan
- Based on the Intel SA-1110 StrongARM processor running at 206 MHz, 64 MB of RAM, and 16MB Flash
- OpenZaurus project started in late 2001 / early 2002 to produce Linux images for the Zaurus, this was the precursor to OpenEmbedded…

(photo courtesy Wikipedia: https://commons.wikimedia.org/wiki/File:Sharp_Zaurus.jpg)

A brief history of ~~time~~size (today)

- Typical embedded system platforms now range up to > 2 GHz CPU clock speeds, RAM sizes of 256 MB to GBs, and flash storage of GBs…
- Barring memory size, even the average modern microcontroller is multiple times more capable than the early Linux embedded targets
- Given this, do we even need to worry about distribution size anymore?

Why bother optimizing distribution size?

- There are embedded products that still benefit from the cost-optimization of reducing RAM and storage footprint
    - e.g. developers interested in using Linux in IoT edge devices
- An increasing need to keep devices up to date means smaller images have a bandwidth and download time advantage, and potentially a reduced security attack surface
- Use cases such as:
    - Small recovery partition images
    - Container images
    - Supporting older hardware (e.g. Tiny Core Linux, http://distro.ibiblio.org/tinycorelinux/)

poky-tiny

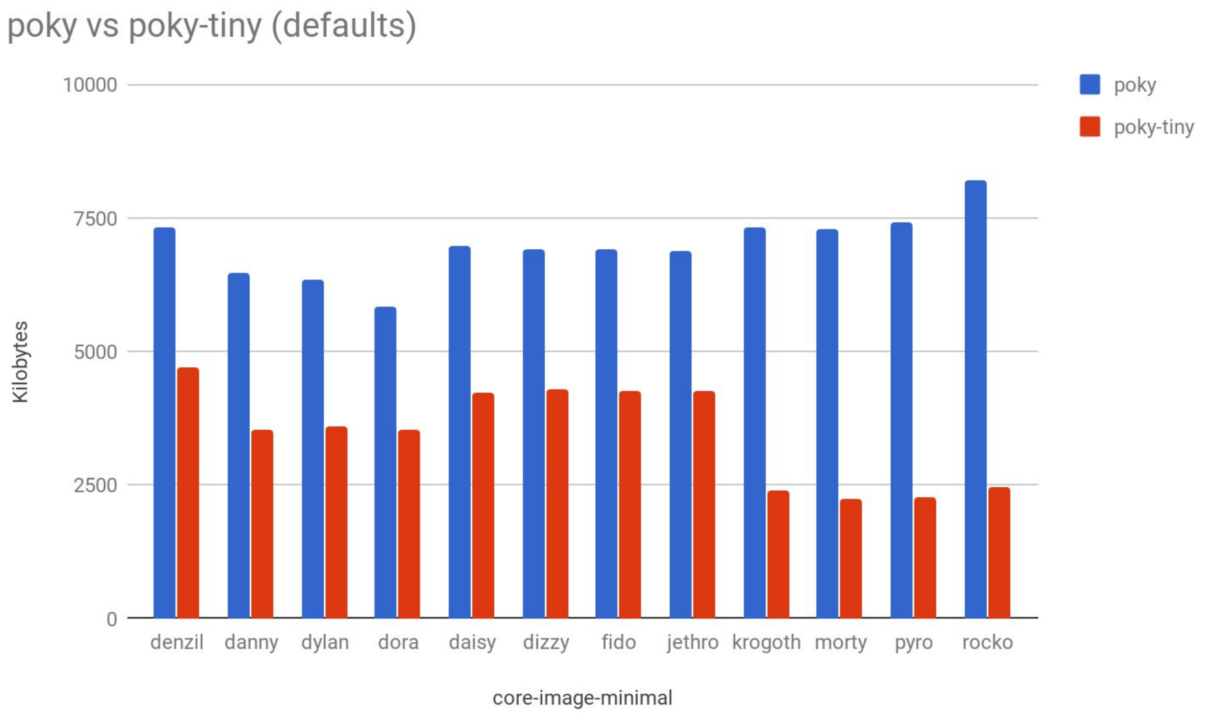
- Added in Yocto Project denzil release in April 2012
- “Poky-tiny is intended to define a tiny Linux system comprised of a Linux kernel tailored to support each specific MACHINE and busybox.” (from poky-tiny.conf)
- As with poky, intended to act as a starting point for your own distribution.
- Caveats:
  - Only builds for qemux86 by default
  - Only supports core-image-minimal
  - So only has a barebones kernel, libc, and BusyBox
- Documented (somewhat) at:
  - https://www.yoctoproject.org/docs/2.4.2/dev-manual/dev-manual.html#building-a-tiny-system

Konsulko Group

poky-tiny contents

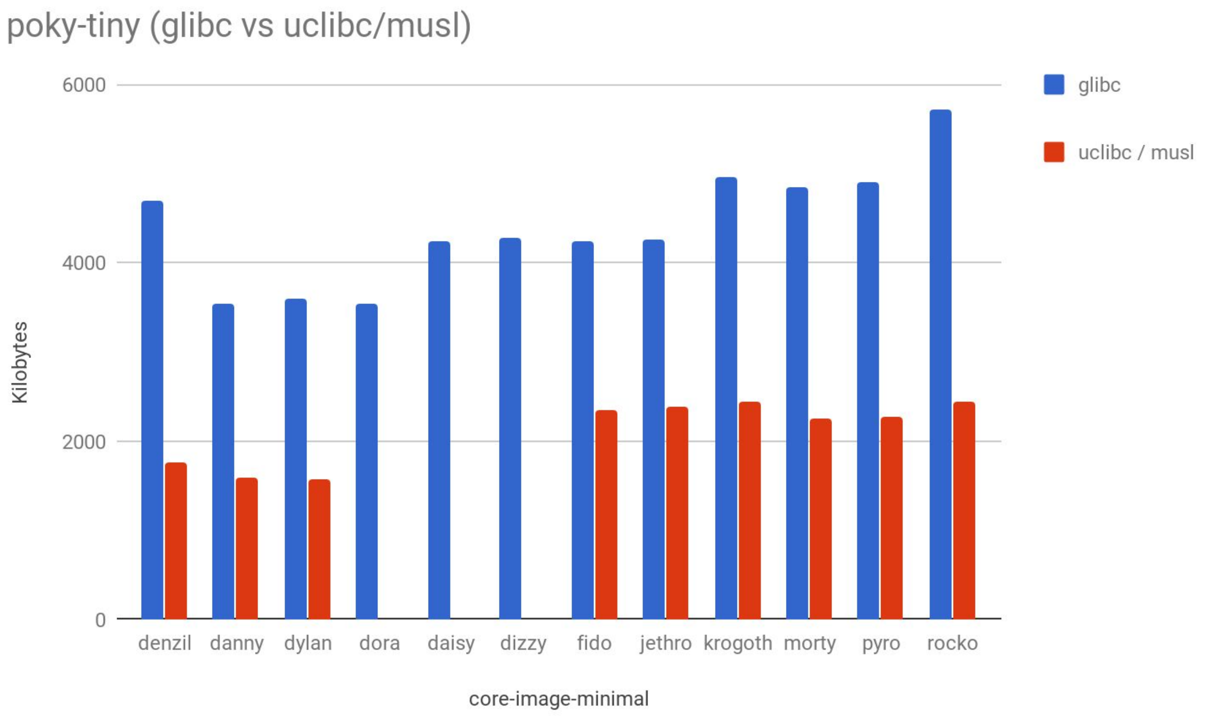
```
$ cat installed-package-sizes.txt
606     KiB     musl
548     KiB     busybox
23      KiB     netbase
4       KiB     update-alternatives-opkg
3       KiB     busybox-udhcpc
3       KiB     busybox-mdev
3       KiB     base-files
2       KiB     run-postinsts
2       KiB     busybox-syslog
0       KiB     packagegroup-core-boot
0       KiB     base-passwd
```

So how big is poky-tiny?



Konsulko Group

So how big is poky-tiny? (continued)



So how big is poky-tiny? (Notes / Observations)

- Numbers pulled from data generated by buildhistory class
- Missing uclibc sizes for dora - dizzy releases due to build issues
- Noticeable trend of glibc increasing in size over time
- No obvious dramatic size difference between uclibc and musl
- Installed package size != root filesystem size
    - There is some overhead due to the inodes from the many small files used by update-alternatives-opkg metadata
    - The root filesystem is assembled from binary packages, and the update-alternatives-opkg tool is used to handle alternate providers of binaries (e.g. BusyBox versus util-linux)
    - Currently there is no configuration to disable installation of its metadata into the image, so removal requires post-processing

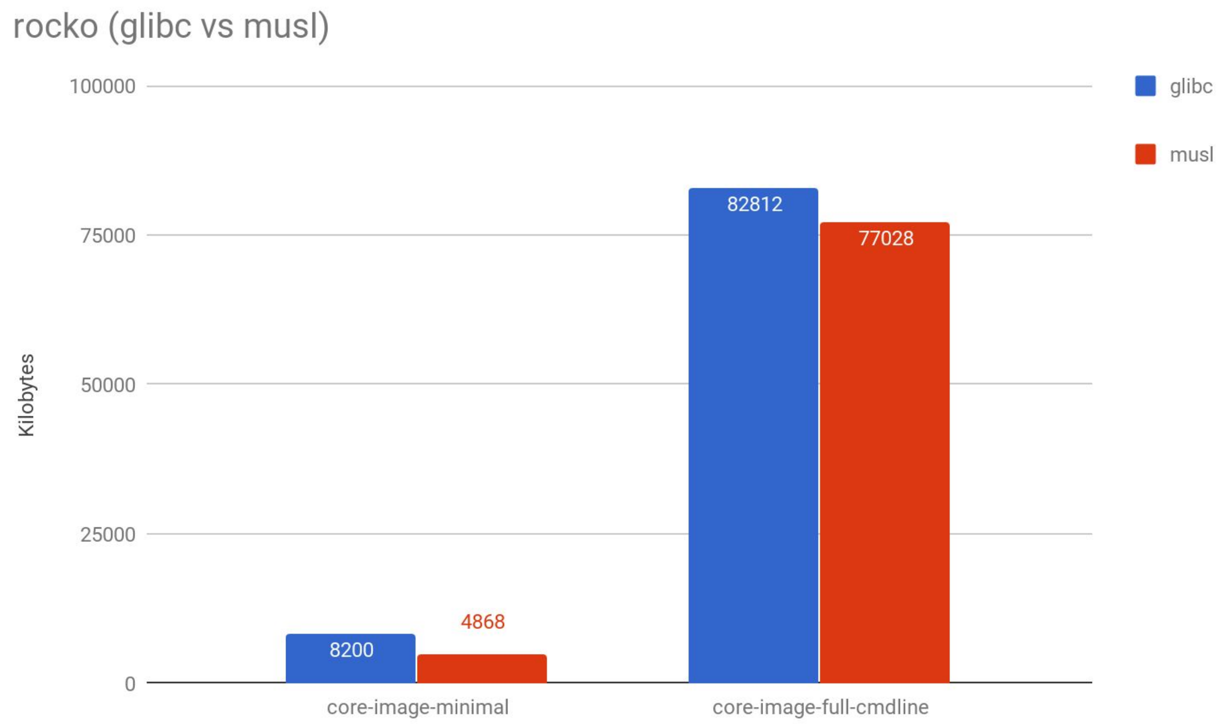
poky-tiny changes versus poky

- TCLIBC
- ENABLE_WIDEC
- USE_NLS
- DISTRO_FEATURES
- linux-yocto-tiny

poky-tiny changes: TCLIBC

- TCLIBC variable selects the standard C library to use; default value is glibc, as of the krogoth release the other option is musl (previously was uclibc)
- musl (https://wiki.musl-libc.org/) is a lightweight C library implementation
  - Actively maintained, MIT licensed
  - In addition to binary size benefits, there are significant runtime memory usage ones as well
- See http://www.etalabs.net/compare_libcs.html for a detailed comparison
- While the recipes in oe-core are test built with musl, recipes from other layers may not work out of the box, and other software may require patching to build
  - Almost all the recipes in meta-openembedded build against musl, the remaining handful are actively being worked on
  - Typical failures are due to accidentally relying on non-standard glibc extension or definitions

How much does musl save?



Konsulko Group

poky-tiny changes: ENABLE_WIDEC

- ENABLE_WIDEC variable controls wide character support for the ncurses terminal library
- Disabling ncurses wide character support only affects console applications
- However, not all applications will build with it disabled
    - core-image-minimal and core-image-full-cmdline images build, core-image-sato does not
- Size savings are not necessarily dramatic
    - About 200 KB if the image pulls in ncurses
    - ATM core-image-minimal does not actually pull in any packages that have ncurses as a dependency...

poky-tiny changes: USE_NLS

- USE_NLS variable controls native language support for applications, i.e. internationalization via the gettext library
- Disabling NLS might be problematic if you have applications using gettext to provide internationalized output
- However, again not all applications will build with it disabled
    - core-image-minimal and core-image-full-cmdline build, core-image-sato does not
- Size savings are dependent on application usage of gettext
    - No savings in core-image-minimal, about 2 MB in core-image-full-cmdline

poky-tiny changes: DISTRO_FEATURES

- DISTRO_FEATURES variable controls software feature support
    - Mostly translates to configure script options, but some features add kernel module and runtime package dependencies
- poky-tiny removes almost all of the default features that poky enables, leaving on IPv4 and IPv6 support on as well as a couple of other base features
- The features you need are largely dependent on your target image contents
    - e.g. x11, pulseaudio, many fine-grained libc features for glibc
- See Chapter 14 of the Yocto Project Reference Manual for a breakdown of DISTRO and MACHINE features
    - https://www.yoctoproject.org/docs/2.4.2/ref-manual/ref-manual.html#ref-features

poky-tiny changes: linux-yocto-tiny

- Provides a highly pruned kernel configuration
- qemux86 specific
- Needs to be over-ridden with PREFERRED_PROVIDER_virtual/kernel if you want to test poky-tiny on another architecture
- General kernel size reduction guidelines apply, i.e. only enable features and drivers for the target platform
- A static kernel without modules is a win if possible since it results in an overall size reduction

Common image feature / package sizes

- Note that size numbers include all the dependencies that are pulled in
- Package management
  - rpm: ~102 MB (includes OpenSSL, Python 3, etc.)
  - deb: ~22 MB
  - ipk: ~4 MB
- SSH daemon:
  - OpenSSH (and OpenSSL): ~6.7 MB
  - Dropbear: ~300 KB
- Systemd: ~30 MB
- Python 2.7: ~4 MB, ~40 MB with all standard modules
- Python 3.5: ~17 MB, ~64 MB with all standard modules

Other size reduction options

- Splitting files out of a package with FILES_${PN}-foo
    - Example use would be to pick out a tool from core-utils, since it is not split into per-tool packages like util-linux
- A more extreme example is removing all .py Python source files, leaving only the compiled .pyc files
    - Currently requires modifying distutils-common-base.bbclass to make it generic for all Python modules
- Use ROOTFS_POSTPROCESS_COMMAND to remove files from the image
    - e.g. removing unneeded update-alternatives-opkg metadata
    - https://www.yoctoproject.org/docs/2.4.2/ref-manual/ref-manual.html#var-ROOTFS_POSTPROCESS_COMMAND
- If using glibc, tweak IMAGE_LINGUAS to remove unwanted locales

Summary / Recommendations

- If starting out, take poky-tiny.conf as a starting point to define your own distribution configuration, then add things to it
- Otherwise, it is likely that switching to using musl will provide the biggest immediate improvement
- Use the buildhistory class to help simplify investigating what is taking up space in your image
  - https://www.yoctoproject.org/docs/2.4.2/ref-manual/ref-manual.html#maintaining-build-output-quality
  - The BUILDHISTORY_COMMIT option allows looking at differences between builds
- For kernel (and BusyBox) size reduction start with the guidelines at:
  - https://www.yoctoproject.org/docs/2.4.2/dev-manual/dev-manual.html#trim-the-kernel

...and Beyond?

- Investigate having a feature to optionally remove alternatives metadata
- There is mention in poky-tiny.conf of an unimplemented “tiny” DISTRO_FEATURE, investigate what that could/should do
  - A scheme to reduce PACKAGECONFIG default settings for large packages with lots of options?
- Take a look at Tiny Core Linux, OpenWRT, and other similar distributions to see if there are tricks that we could emulate
  - Perhaps a project to attempt to produce a distribution similar in content to Tiny Core, to see if that reveals where any further issues lie
- Container images

Questions?