Bootstrapping Debian-based distributions for new architectures

Johannes 'josch' Schauer

Jacobs University Bremen

FOSDEM 2013, Brussels

Overview

- Started as Debian Google Summer of Code project 2012
- Continued as my master thesis at Jacobs University Bremen
- Mentors:
  - **Wookey** practical side of things
  - **Pietro Abate** theoretical and academic side of things

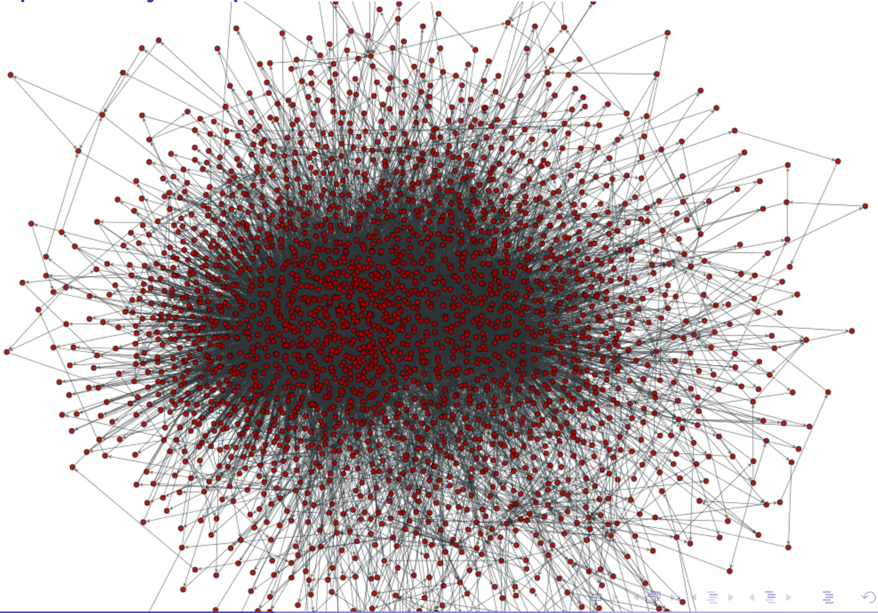
Assumptions

- Source packages are always natively compiled
- Source packages are compiled with the full archive of binary packages available

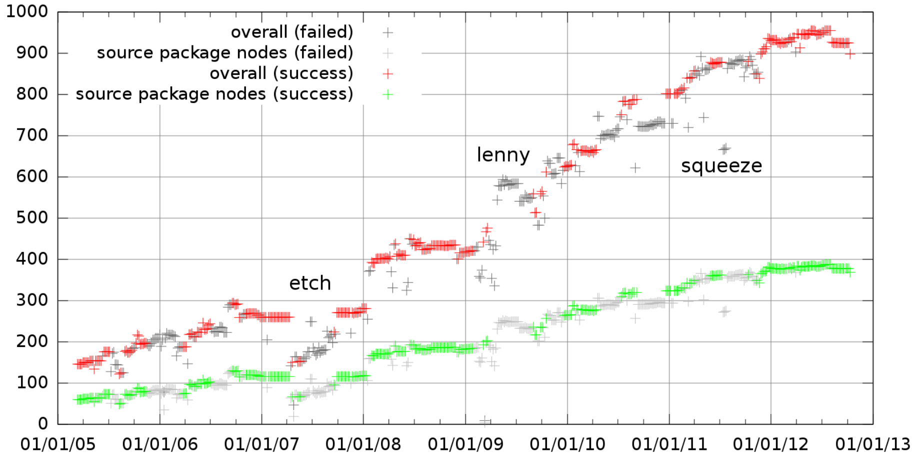
During Bootstrapping

- Some source packages must be cross compiled
- Only a few binary packages are available $\rightarrow$ dependency cycles

Dependency Graph in Debian Sid



Development of problem size



Current Bootstrapping Practice

- Using Gentoo or OpenEmbedded to avoid cross compilation
- Manual dependency cycle analysis
- Manual hacking of source packages to build with fewer build dependencies
- Takes up to a year to complete

How needed bootstrapping is

- 20 Debian ports so about 1 port per year

What if bootstrapping was easier

- Porting for upcoming architectures easier and faster
- More subarch builds, optimized for a specific CPU

What else would be nice

- Remove the need of Gentoo or OpenEmbedded
- Update lagging architectures
- QA tool which allows to check the archive for bootstrappability

The essence of this talk

- We now have the algorithms to automatically do all this

The tools

- Written in OCaml, Python, Shell
- Using dose3 as helper library
- Git: `https://gitorious.org/debian-bootstrap/bootstrap`
- Code contributions by Pietro Abate

More specifically we can now...

- ... create & analyze a dependency graph
- ... find source packages to modify
- ... create a build order

It's only theory

- Tools only work on package meta data
- No source packages are compiled, no binary packages installed
- Ignoring the practical implementation of cross compilation, reduced build dependencies

What is needed to test in practice

- More multiarch (cross compilation)
- Better cross compilation support in base packages
- Reduced build dependencies (build profiles)

Cross compilation

1. Select packages for minimal native build system
   - `essential:yes`
   - `build-essential`
   - `debhelper`
2. Get their co-installation set
3. Get their source packages

Algorithm: select packages to cross compile

1. Add source packages to result
2. Get foreign cross build dependencies
3. Get source packages to build them
4. If not in result go to (1)

Multi-Arch conflicts

- Cannot resolve cross build dependencies of some source packages due to `Multi-Arch` conflicts
- Ignoring cross for now
- Assumption: minimal native build system can be created from nothing
- Later: building and solving dependency graph just as in native phase

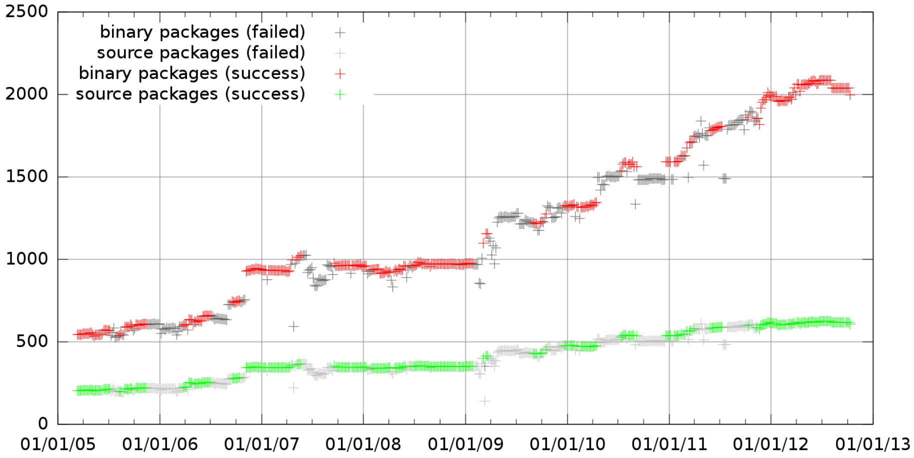
Native compilation

- Starting from minimal native build system, create and analyze a dependency graph
- Debian is huge (18k source, 38k binary)
- Create reduced distribution first

Reduced distribution

- A selection of source packages and binary packages:
  - All binary packages must be created by the source packages
  - All source packages are compilable

Development of reduced distribution size



Selecting packages for a reduced distribution

1. For example:
   - `essential:yes`
   - `build-essential`
   - `debhelper`
2. Get their co-installation set
3. Get their source packages

Algorithm: select packages for reduced distribution

1. Add source packages to result
2. Get build dependencies
3. Get source packages to build them
4. If not in result go to (1)

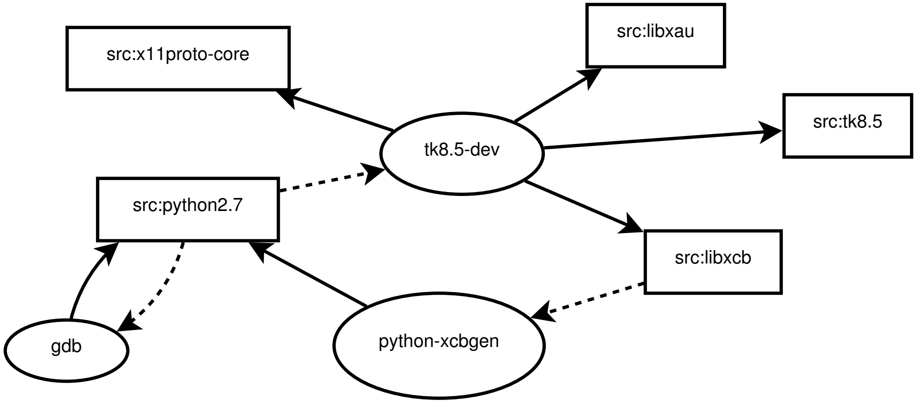
Build graph

- A dependency graph
- Vertices
  - Source packages
  - Installation sets
- Edges
  - Build-depends (source $\rightarrow$ installation sets)
  - Builds-from (installation set $\rightarrow$ sources)

Building the graph

- Connect source packages to installation sets of their build dependencies (except installable ones)
- Connect installation sets to source packages they build from (except available ones)

Example build graph

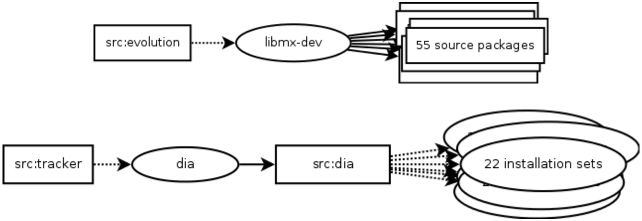


Breaking dependency cycles

- Remove build dependencies (build profiles)
- Move dependencies from `Build-Depends` to `Build-Depends-Indep`
- Choose different installation sets for not-strong dependencies
- Make binary packages available through cross compilation

Finding source packages to modify

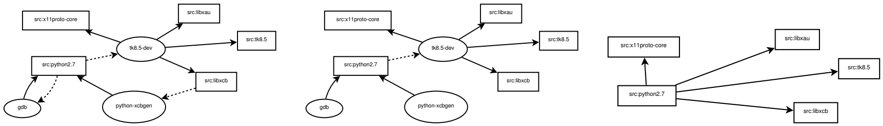
- Least amount of build dependencies missing
- Ratios



- List small cycles
- Edges part of most cycles
- Calculate Feedback Arc Set

Creating a build order

- Feedback vertex set problem: find a small amount of source packages to profile build and make build graph acyclic
- Convert build graph into source graph



- Topologically sort source vertices
  1. `src:python2.7(*)`
  2. `src:x11proto-core`, `src:libxau`, `src:tk8.5`, `src:libxcb(*)`

Demo time!

- Debian Sid 1. January 2013
- reduced distribution: 613 source packages, 2044 binary packages in 75 seconds
- full distribution: 18613 source packages, 38433 binary packages in 9 minutes
- credit for reduced build dependencies goes to Gentoo, Thorsten Glaser, Patrick McDermott, Daniel Schepler, Wookey

TODO

- Try it out in real life
  - More multiarch
  - More cross compilation
  - Decide for a build profile syntax & field names
  - Implement build profiles
- Better heuristics
- Generalize for larger problem class
- Finding a name

Resources

- Blog: `http://blog.mister-muffin.de`
- ML: `debian-bootstrap [at] lists.mister-muffin.de`
- IRC: `#debian-bootstrap [at] irc.oftc.net`
- Git1: `https://gitorious.org/debian-bootstrap/bootstrap`
- Git2: `https://gitorious.org/debian-bootstrap/gen2deb`
- Git3: `https://github.com/josch/cycle_test`
- Dose3: `https://gforge.inria.fr/projects/dose/`
- Wiki1: `http://wiki.debian.org/DebianBootstrap`
- Wiki2: `http://wiki.debian.org/DebianBootstrap/TODO`
- Profiles: `https://l.d.o/debian-devel/2013/01/msg00329.html`

Questions

Questions?