# Dependency Evolution Analysis on Arch Repositories

By Wenhan Zhu (Cosmos)

## Related work

Previous work done on dependency network analysis for NPM, Cargo, Gem [1]

Installability analysis base on dependency for Debian, OPAM, CRAN [2]

Work done on dependency network analysis of PyPI, CRAN, NPM show that each ecosystem are not generalizable [3]

# Why Arch?

# Arch Linux

- Dependency Network evolution
- Installability Issues

| | **Monitored** | **Maintained** |
|---|---|---|
| Language | NPM, PyPI, Gem, Drupal, CRAN | OPAM |
| Distribution | **Arch User Repository (AUR)** | Debian |

For all large scale distributions, Arch has the only monitored repository (AUR).

# Terminology

### Software repository

A software repository, colloquially known as a "repo" for short, is a storage location from which software packages may be retrieved and installed on a computer.[Wiki]

### Dependency

```
A --> B --> C
```

If A depends on B and B depends on C.

- B is a dependency for A
- A is a dependent of B
- C is a transitive dependency for A
- A is a transitive dependent for C

# Arch Repositories

## Official

2 separate SVN repositories that holds all packages in each. Have a git mirror.

- Core and Extra (Refer as Core from now on)
- Community

## Unofficial

Each package hosted on separate Git repos.

- Arch User Repository (AUR)

# Arch Repository Structure

## Official

```
file
|__repos
|  |__architecture
|       |__PKGBUILD
|__trunk
   |__PKGBUILD
```

## AUR

Each git repo

```
file
|__PKGBUILD
|__.SRCINFO
```

## Port-like System

Build script that contains every information to build the software.

**PKGBUILD**: is a shell script containing the build information required by Arch Linux packages.

## Metadata for *PKGBUILD*

*.SRCINFO* extracted from *PKGBUILD*

## Structure of official

`repos/architecture/PKGBUILD` contains the code that builds the binary.

`trunk/PKGBUILD` is for development space.

# Examples

## *PKGBUILD*

```
pkgname=glibc
pkgver=2.26
pkgrel=4
pkgdesc='GNU C Library'
arch=(i686 x86_64)
url='http://www.gnu.org/software/libc'
depends=('linux-api-headers>=4.10' tzdata filesystem)
makedepends=(git gd)
optdepends=('gd: for memusagestat')
...
```

```
.right-column[
```

## *.SRCINFO*

```
pkgbase = glibc
    pkgdesc = GNU C Library
    pkgver = 2.26
    pkgrel = 4
    arch = x86_64
    makedepends = git
    makedepends = gd
    depends = linux-api-headers>=4.10
    depends = tzdata
    depends = filesystem
    optdepends = gd: for memusagestat
    options = !strip
    options = staticlibs
...
pkgname = glibc
```

# Cloning

## Official

Clone the git mirror of svn repositories.

First commit:

- Core: 2008 Apr. 6
- Community: 2009 Jul. 16

## AUR

AUR hosts a list of all currently available packages. Used a tool `auracle` to clone each repo.

4000 API limits per IP per day

**Problem**

- Correctness of Tool
- No way of getting deleted packages

# Extract *.SRCINFO*

## Official

Does not contain *.SRCINFO* file. Created VM to run command to extract *.SRCINFO*.

Packages that does not generate *.SRCINFO* file or does not follow standard file structure are ignored.

## AUR

Parse *.SRCINFO* file given.

## Validation of parsed results

Compare with Arch official website Followed Arch wiki for standard

## Snapshots

Took snapshots by 3 months from 2016 Mar. to 2018 Sept.

## Dependency

Compute as directed graph.

- # of depends: Size of child of each node
- # of transitive depends: Size of transitive enclosure of each node

# Analysis

## Installability

- Dependency conflict

```
A -> B.ver == 1
A -> C
C -> B.ver == 2
```

- Dependency cycle
- Dependency missing

```
A -> B

B not in repository
```

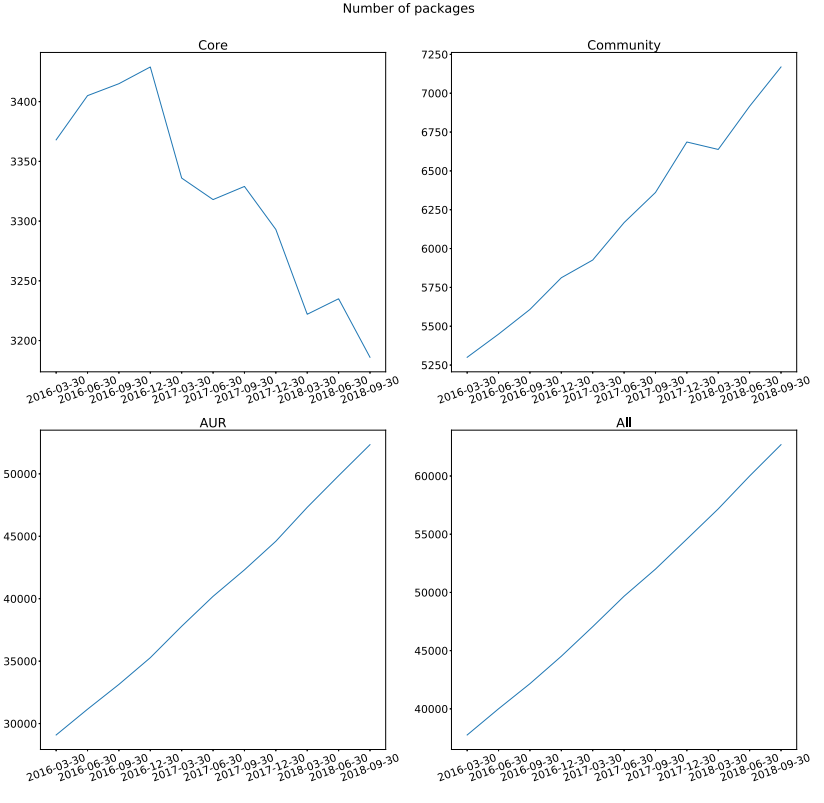
Dependency cycle could happen in arch official repositories!

## Example

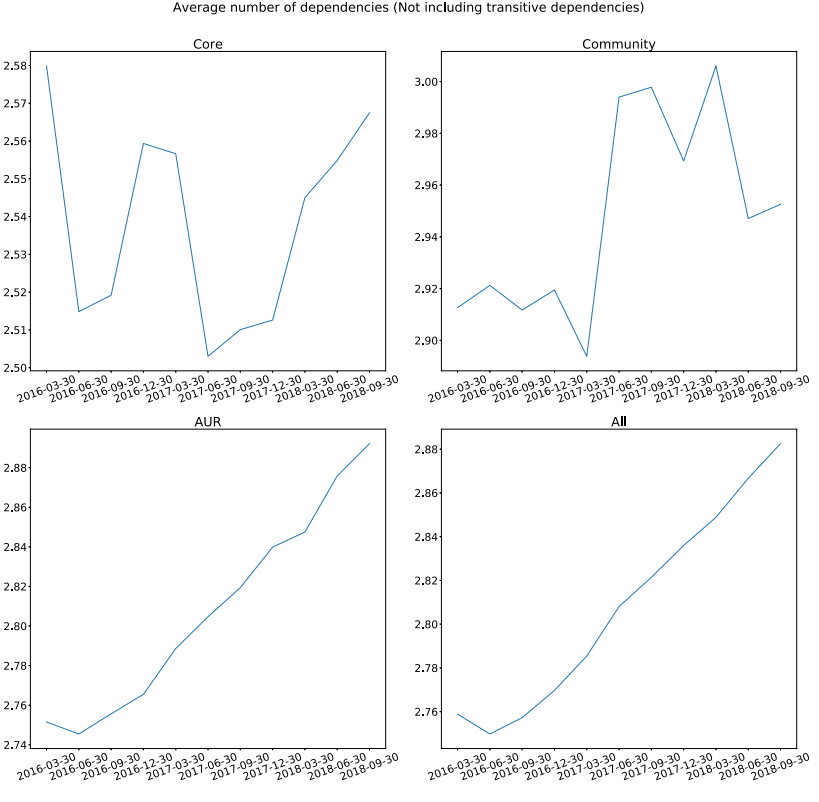
Cycle of dependency

```
freetype2 <--> harfbuzz
```

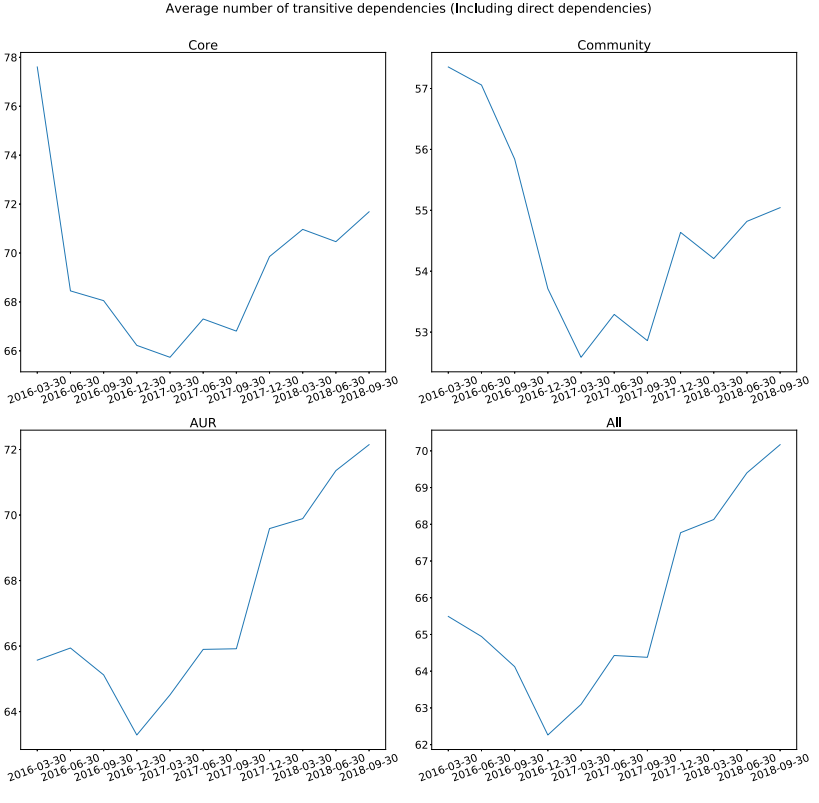
# Results


Number of packages
Core
Community
AUR
All
3400
3350
3300
3250
3200
7250
7000
6750
6500
6250
6000
5750
5500
5250
50000
45000
40000
35000
30000
60000
55000
50000
45000
40000
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

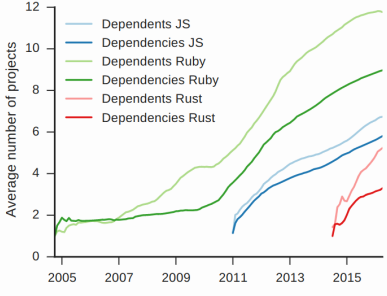
# Results


Average number of dependencies (Not including transitive dependencies)
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

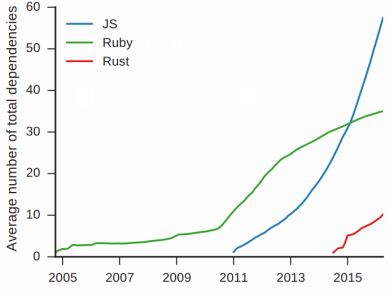
# Results


Average number of transitive dependencies (Including direct dependencies)
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

# Comparison

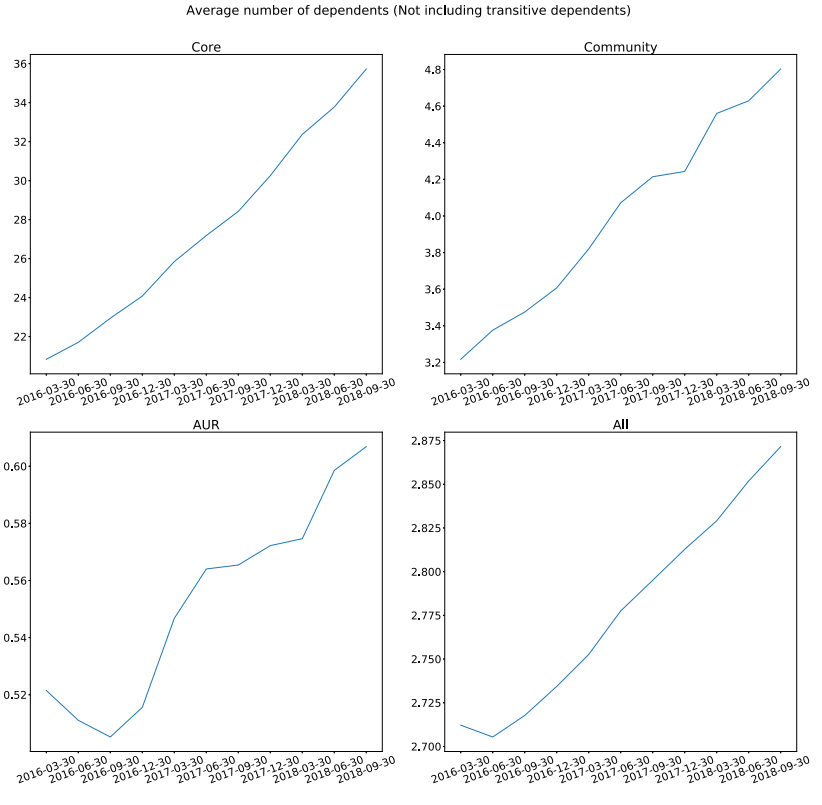



(a) Evolution of the number direct dependencies and dependents for each project version. Average over those that have at least one dependency or one dependent.

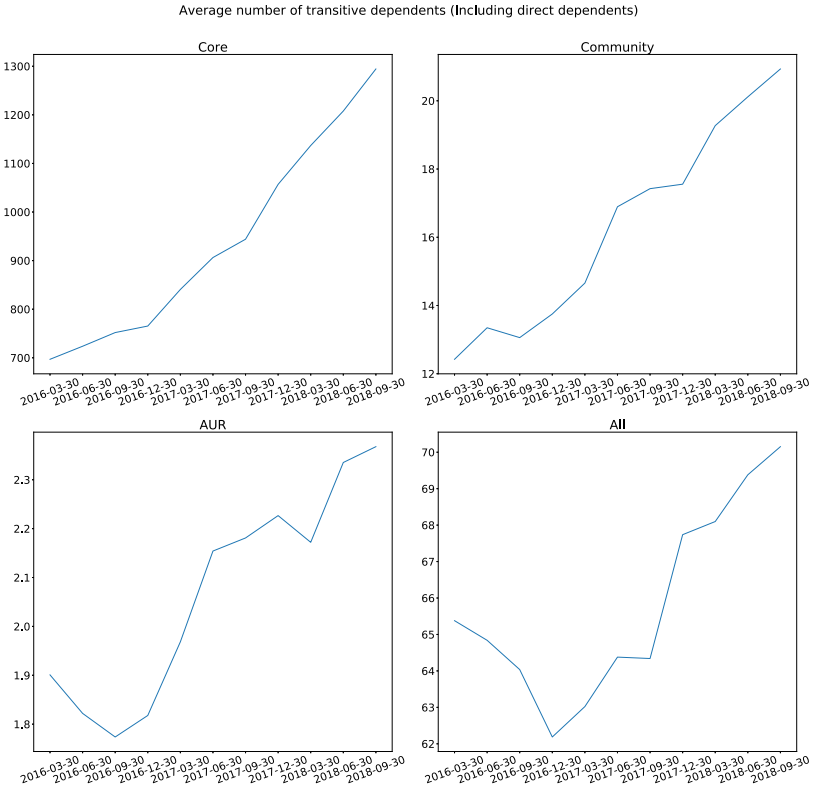



(b) Average number of total dependencies, including the full transitive closure. Average calculated over projects that have at least one dependency.

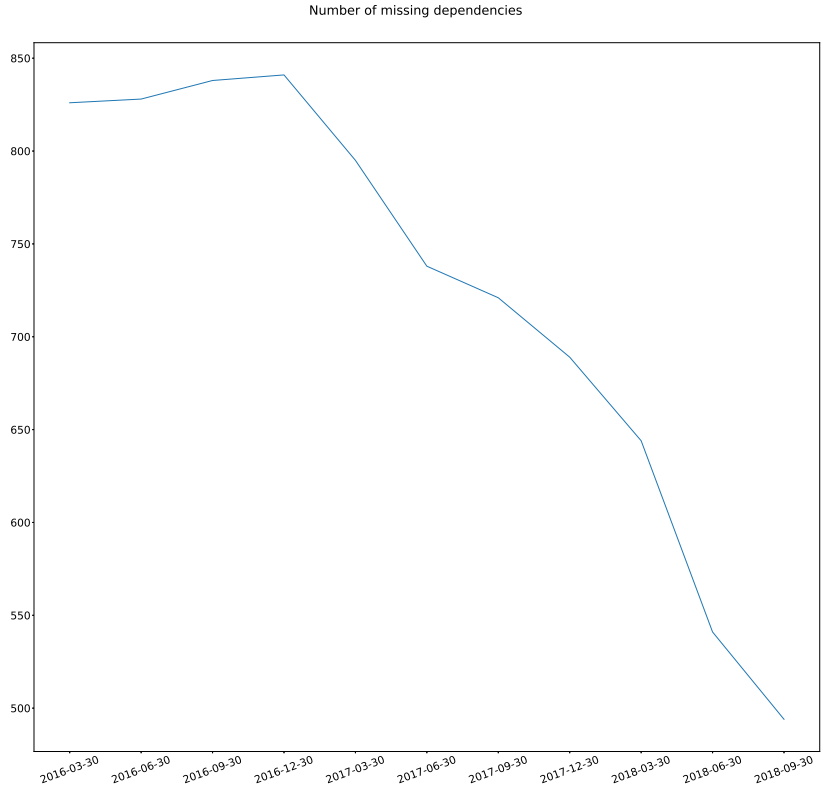
# Results


Average number of dependents (Not including transitive dependents)
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

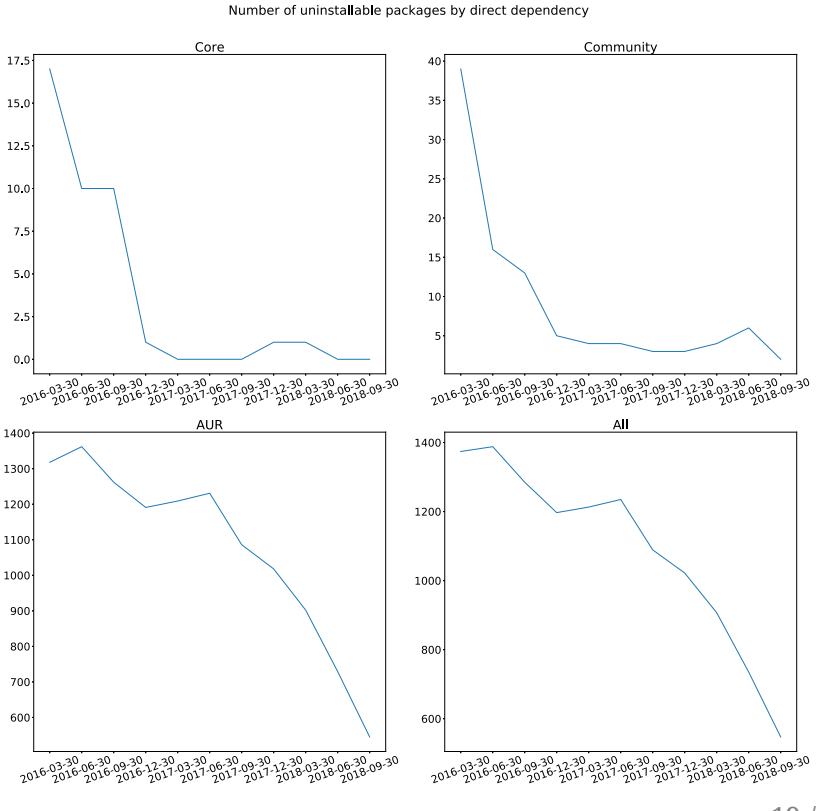
# Results


Average number of transitive dependents (Including direct dependents)
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

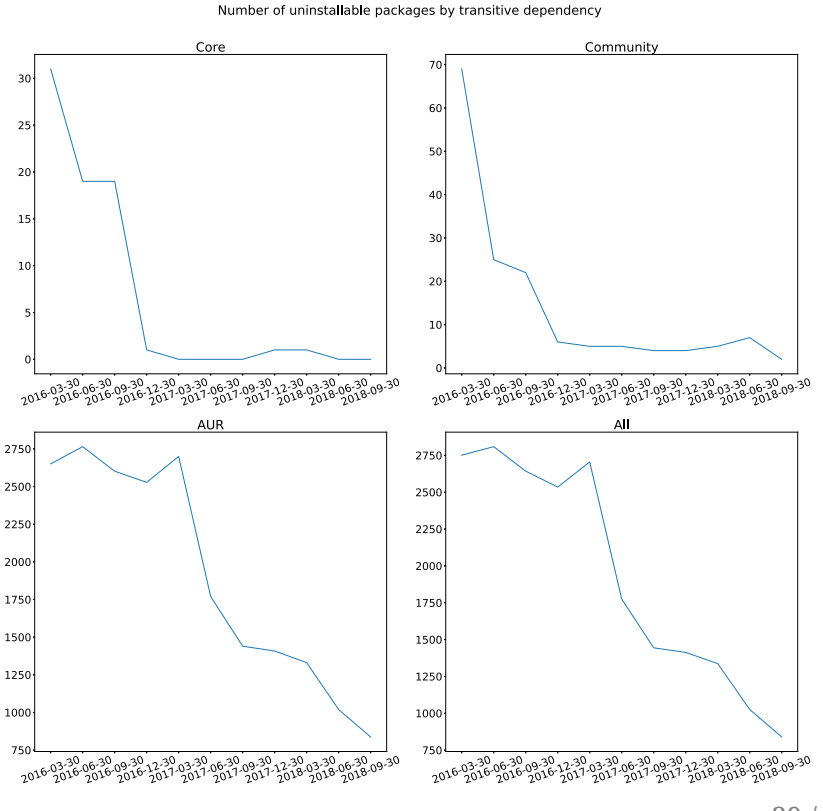
# Results


Number of missing dependencies
850
800
750
700
650
600
550
500
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

# Results


Number of uninstallable packages by direct dependency
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

# Results


Number of uninstallable packages by transitive dependency
Core
Community
AUR
All
2016-03-30
2016-06-30
2016-09-30
2016-12-30
2017-03-30
2017-06-30
2017-09-30
2017-12-30
2018-03-30
2018-06-30
2018-09-30

# References

[1] Structure and evolution of package dependency networks. *Riivo Kikas, Georgios Gousios Marlon Dumas Dietmar Pfahl*. MSR '17

[2] Mining component repositories for installability issues. *Pietro Abate, Roberto Di Cosmo, Louis Gesbert, Fabrice Le Fessant, Ralf Treinen, Stefano Zacchiroli*. MSR '15

[3] On the topology of package dependency networks: A comparison of three programming language ecosystems. *A. Decan, T.mens, M. Claes*. ECSAW 2016

# Thank you!

Slides powered by Remark