Using a time dependency graph to find the most widely used Debian packages

AUTHOR
Teodor Dobrev
T.Dobrev@student.tudelft.nl

AFFILIATIONS
Supervisor: Georgios Gussios
G.Gussios@tudelft.nl

Superviosr: Diomidis Spineellis
D.Spinellis@tudelft.nl

1 BACKGROUND

- Reusing already existing packages introduces a number of dependencies in an application.
- Once an exploit is found in any of these dependencies, the whole application becomes vulnerable. Some famous examples are the Log4J and the leftPad cases.
- Existing work does not include the transitive dependencies based on the time of the release of a package. (Fig. 1)

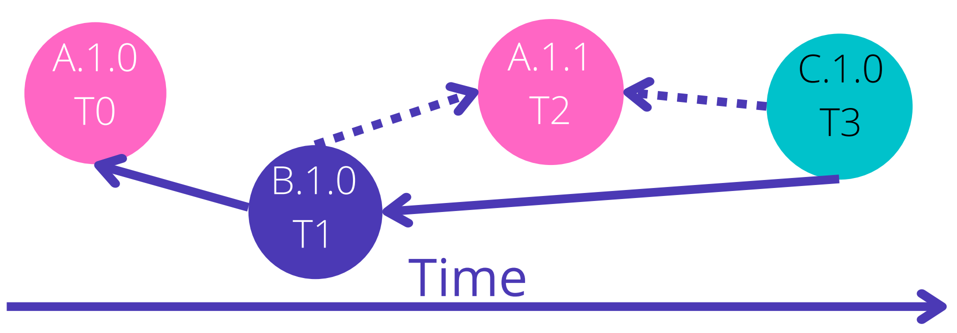

Figure 1: The dependencies between 3 packages (A, B and C) in time (denoted as T)

2 RESEARCH QUESTIONS

Which are the most widely used Debian Packages?

- How to efficiently design a dependency graph structure?
- How does time influence a dependency graph structure?
- What measures of criticality could be used to evaluate this graph data structure?

3 METHODOLOGY

- Collecting the Debian data from the Snapshot Archive [1]
- Creating a timed graph structure and mapping the data onto it. This is done by introducing an individual node per version, timestamped with its release date.
- Comparing querying of the graph with the resolutions provided by the package manager.
- Running a PageRank algorithm to find the most widely used packages.

4 RESULTS

- Fig. 2 provides an overview of the nodes with the highest page rank scores. Those were defined as the most widely used packages in the context of this research.
- Fig. 3 shows that the implemented time dependent graph structure actually resolves dependencies according to each version. The results are consistent with those in Fig. 2, considering that there they are aggregated.
- In Fig. 4 can be observed the popularity of the packages over the years. It can be noticed that libc6 and libgcc1 are closely related, apart from 2022 when the libgcc-s1 is released.

Package	PageRank
libc6	0.2811
libgcc1	0.1906
multiarch-support	0.0387
libgcc-s1	0.0210
libcrypt1	0.0206
libjs-jquery	0.0132
gcc-6-base	0.0087
perl	0.0072
dpkg	0.0070
libuima-core-java	0.0069

Figure 2: PageRank results aggregated by package (Average of 10 runs)

Package	Version	PageRank	Release Date
libgcc1	1:8.3.0-6	0.042894	2020-01-01
libgcc1	1:6.3.0-18+deb9u1	0.042891	2018-06-01
libgcc1	1:6.3.0-18	0.042891	2017-06-18
libgcc1	1:4.9.2-10	0.042885	2015-06-01
libgcc1	1:4.7.2-5	0.042885	2015-01-01
libc6	2.31-13+deb11u3	0.026273	2022-06-01
libc6	2.31-13+deb11u2	0.026273	2022-01-01
libc6	2.28-10	0.025670	2020-01-01
libgcc-s1	10.2.1-6	0.024291	2022-01-01
libcrypt1	1:4.4.18-4	0.023923	2022-01-01

Figure 3: PageRank results per individual version (Average of 10 runs)

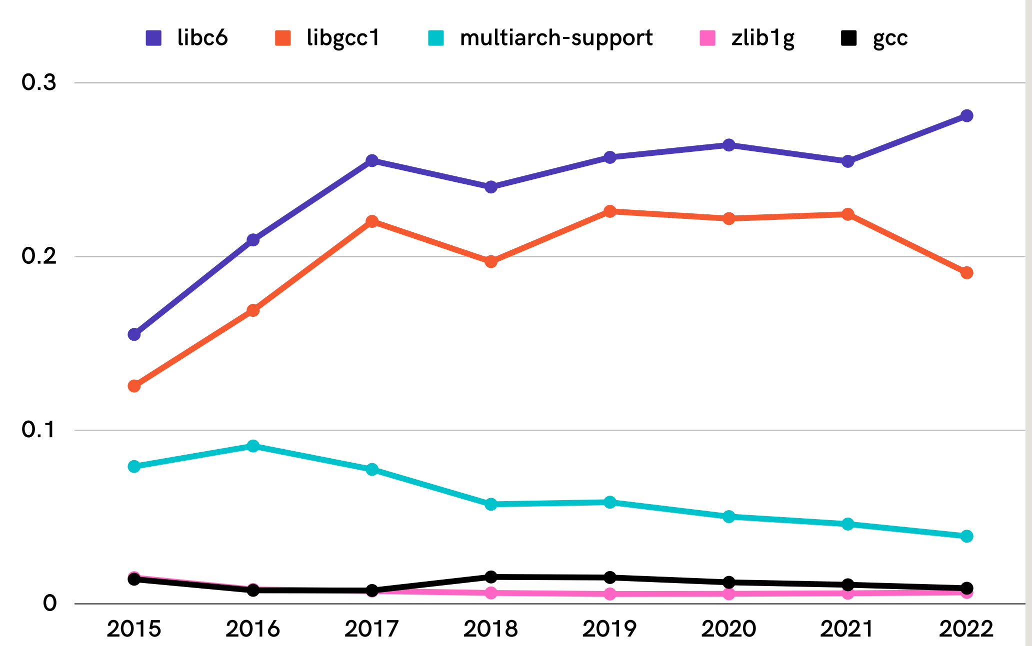

Figure 4: PageRank results over time (Average of 10 runs)

5 CONCLUSION & FUTURE WORK

- Debian packages were mapped onto a time dependency graph. It resolved their dependencies as accurate as the package manager itself.
- Introducing the time component increased the precision of dependency resolving as it can be seen in Fig. 3.
- A point of improvement is the scalability of the graph since it does not perform well enough on larger datasets.
- Currently, this research does not show the benefits of improving the precision of the dependency resolving.

RELATED LITERATURE

[1] https://snapshot.debian.org/