A Quantitative Assessment of Package Freshness in Linux Distributions

Damien Legay
Software Engineering Lab
University of Mons
Mons, Belgium
damien.legay@umons.ac.be
ORCID 0000-0001-6811-6585

Alexandre Decan
Software Engineering Lab
University of Mons
Mons, Belgium
alexandre.decan@umons.ac.be
ORCID 0000-0002-5824-5823

Tom Mens
Software Engineering Lab
University of Mons
Mons, Belgium
tom.mens@umons.ac.be
ORCID 0000-0003-3636-5020

***Abstract*—Linux users expect fresh packages in the official repositories of their distributions. Yet, due to philosophical divergences, the packages available in various distributions do not all have the same degree of freshness. Users therefore need to be informed as to those differences. Through quantitative empirical analyses, we assess and compare the freshness of 890 common packages in six mainstream Linux distributions. We find that at least one out of ten packages is outdated, but the proportion of outdated packages varies greatly between these distributions. Using the metrics of update delay and time lag, we find that the majority of packages are using versions less than 3 months behind the upstream in 5 of those 6 distributions. We contrast the user perception of package freshness with our analyses and order the considered distributions in terms of package freshness to help Linux users in choosing a distribution that most fits their needs and expectations.**

I. Introduction

Since its inception in 1991, the Linux operating system has continued to expand and evolve. Its kernel has grown at a superlinear rate [1], [2]. Since Linux uses the very permissive open source license GNU GPL, it has been forked into a myriad of variants, called *distributions*. These distributions are often built around a *package manager* that complements the Linux kernel with a plethora of third-party software *packages*. Much of the functionality provided by a distribution to its end-users comes from these third-party packages. They are provided to users via distribution-specific *software repositories*. As a result, Linux distributions form *software ecosystems* in which packages interact in complex relationships of dependency, co-installability [3], redundancy and complementarity. As such, the versions of packages available in the official repositories (repositories that are enabled by default) of different distributions may vary according to the philosophy adopted by the creators and maintainers of the distribution, as they take different approaches to nurturing the health of their distributions.

Some maintainers place great emphasis on maximising the *stability* of the distribution, to avoid the risk of introducing changes that could potentially introduce package incompatibilities or break prior functionality. A distribution known to markedly emphasise stability is the Stable branch of Debian. Some maintainers put security concerns at the forefront, aiming to make their distributions as resistant as possible to nefarious acts. One such distribution is Qubes OS [4], which minimises the potential impact of security vulnerabilities by isolating software components as much as possible through the use of virtual machines. Other maintainers, such as those of Arch Linux, prioritise package *freshness* by endeavouring to incorporate package updates as quickly as possible.

In prior work [5], we conducted a qualitative survey of 170 Linux users revealing that Linux users consider package freshness important, as package updates are a source of security patches, bug fixes and new features. This is also attested by the existence of package freshness monitoring services such as Repology and DistroWatch. The survey also highlighted that users tend to rely on their distributions' official repositories to install and update packages. Therefore, it is important for users to be well-informed as to the relative freshness of the packages in various Linux distributions. The survey examined the user perception of package freshness in the official repositories of the distributions, observing vast discrepancies between distributions, from users of Arch expecting package updates to be deployed within days, to users of Debian Stable and CentOS deeming that it would take months. The survey also established that whenever fresh versions of packages are not available within the official repositories of the distribution, users have to use other means of updating, which can be detrimental to their system's stability and security. Therefore, in order to evaluate to which extent user perception matches objective quantitative evidence and to enable them to make an informed choice of distribution, we empirically measure and compare the package freshness of six popular distributions: Arch Linux, CentOS, Debian Stable, Debian Unstable, Fedora and Ubuntu, over a period of 5 years. We do so by studying three research questions for each distribution. RQ_1: How prevalent are outdated package versions in Linux distributions? RQ_2: How long have those package versions been outdated? RQ_3: To which extent are deployed packages outdated?

II. Related Work

The notion of freshness has been studied in non-Linux environments. Cox et al. [6] proposed system-level metrics to quantify a software system's *dependency freshness*, that is to

say how up-to-date the system's dependencies are. Gonzalez-Barahona et al. [7] introduced the concept of *technical lag* as a measure of how outdated a system is with respect to its dependencies. They defined the technical lag in terms of a lag function and lag aggregation function for packages. This notion was generalised further and used by Zerouali et al. [8].

Aspects of package co-installability and package dependency in Linux have been well studied, particularly with regards to Debian. Vouillon et al. [3], [9] and Claes et al. [10] examined the co-installability problem as it applied to the Debian ecosystem and how it could be solved. Similarly, Artho et al. [11] proposed detection and prevention strategies for this problem. Galindo et al. [12] characterised the relationships between packages within Debian, proposing a language to describe whether a package depends on another, has co-installability issues with another or provides the same functionality or a superset of the functionality of another. Nguyen and Holt [13] studied the life cycle of Debian packages. They compared the age of packages in Debian Stable, Unstable and Testing, defining package age as the time delta between its introduction into the distribution and its removal from Debian or its update to a newer version.

There has been little focus on the freshness of packages in Linux distributions. Gonzalez-Barahona et al. [14] observed in 2009 that one out of eight packages within Debian Stable (12%) was not updated at all during a nine-year timespan, from Debian Stable 2.0 (released on 1998-07-24) to 4.0 (released on 2007-04-08). The work most closely related to this paper is a 2009 Bachelor's thesis by Shawcroft [15] comparing the freshness of packages in 8 Linux distributions (Arch Linux, Debian, Gentoo, Fedora, OpenSUSE, Sabayon, Slackware and Ubuntu). The thesis reports that around 20% of the packages in Arch Linux are outdated and from 40% to 60% in the other distributions, and that Debian and Slackware packages were, on average, more obsolete than those of the other distributions. Although his analysis concerns 8 Linux distributions, it only covers 137 packages.

III. Methodology

This section presents the methodology for our empirical study, for which a replication package is available on Zenodo[1].

In order to study package freshness in Linux distributions, we need to select a set of relevant distributions. According to Distrowatch, there are 913 Linux distributions, 274 of which are considered "active". Not all distributions fall within the scope of this study (e.g. Bicom System is a distribution whose sole purpose is to serve as a telephony platform). We focus on general-purpose GNU-based distributions, and select Arch Linux, Debian, CentOS, Fedora and Ubuntu. These distributions were found to be used by 81% of respondents in our survey of Linux users [5].

These distributions have different release policies. Most of them use point releases, in which a new version of the distribution is released at regular intervals. Ubuntu has a fixed release cycle with six-month intervals, releasing in April and October. Fedora also releases two versions a year, but on a looser schedule. CentOS is based on the source code of Red Hat Entreprise Linux (RHEL). For example, CentOS 6.5 is based on the fifth update of the sixth release of RHEL. Arch Linux follows a rolling release policy, wherein packages are constantly updated. There are therefore no explicit version releases of Arch Linux. Debian includes several distributions. Debian Stable is the officially recommended distribution. Debian Testing and Debian Unstable are development branches: packages updates and new packages start in the Debian Unstable rolling release and are continuously updated. When a given package or update fulfils certain requirements, it is moved to Debian Testing. Every 18 months, Testing is frozen, only receiving critical fixes from Unstable. Six months later, a new release is made: Testing becomes Stable and is unfrozen. The previous Debian Stable becomes Debian Oldstable.

To conduct empirical analyses on the freshness of packages in the selected distributions, we require data on the package versions contained within them. We relied on the repositories and archives of official distributions to obtain this data. For distributions relying on point releases, we gathered data on the package versions present in the distribution at the moment of release. For distributions relying on rolling releases, we gathered data on daily snapshots of the distributions. We focused the analysis on a five-year observation period: $[2015, 2020[$. We therefore selected the distribution releases corresponding to that period. They are, respectively: Fedora 23 to 31, Ubuntu 15.04 to 19.10 and CentOS 7.1 to 7.7. We selected CentOS 7 over CentOS 8 for the analysis because CentOS 8 only had one release during the observation period (8.0, released on 2019-09-24). For Debian, we included Stable 8 to 10 and daily snapshots of Unstable. We did not include Testing because, at the moment of release, it is identical to Stable.

To be able to compare distributions, we needed to select a set of packages that are common to all distributions, i.e. packages that are present in at least one snapshot of each distribution. Yet, different distributions might not adopt a package under the same name. For example, Xephyr X-server is available in Debian-derived distributions as the xserver-xephyr package and as xorg-server-xephyr in Arch Linux. To take into account these package name variations across distributions, we established a mapping between the names of packages in the selected distributions. As Arch Linux was found to have the lowest number of packages (from $\approx$ 7k in 2015 to $\approx$ 10.5k by 2020), we mapped the names of packages found in Arch Linux to those found in other distributions. Manually looking at all possible pairs of distinct package names would have been overly time-consuming (there are up to 62k packages in recent releases of Ubuntu and Debian), we thus first computed the normalised Levenshtein edit distance between each pair of package names, only retaining as mapping candidates those whose distance was below 0.25 from each other. The mapping candidates were then grouped into sets of package names by agglomerative clustering. Finally, we manually examined all remaining 6,639 sets of names

[1] http://doi.org/10.5281/zenodo.4446468

to determine which of them actually correspond to the same package. After this mapping process, we had identified 1,065 packages common to all our distributions.

The versioning schemes of 124 of those packages could not be automatically compared between distributions. For instance, the versions of package lua-socket are presented in the form of version numbers in most distributions, but as dates in Arch. With no available data source allowing us to determine equivalence between versioning schemes, we had to exclude these packages. To compare the versions of packages in the distributions, we used libversion[2], a library that takes into account common versioning notation and keywords. For instance, libversion would order the following version numbers as such: $1.0rc1 < 1.0 < 1.0patch1 < 1.1$. We finally excluded 51 packages for which a single version appears in all the selected distributions over the observation period, as it is not meaningful to compare the freshness of such packages. The final dataset contains 890 packages. As these packages are common to the distributions, we avoid distribution-specific packages and cover widely-used packages, including, for instance gcc, zsh, nodejs and gimp.

Since we aim to measure the freshness of packages within Linux distributions, we need to be able to determine when a package version was released by its maintainers. We call this the *upstream release date*, as opposed to the dates at which the package version is deployed *downstream* into various Linux distributions. Unfortunately, we found no aggregate source of information on upstream release dates. Using the official sites of the packages themselves is impractical, as these are all formatted differently, potentially requiring a custom script for each package to extract the data. As a result, we decided to use the date of first appearance of package versions in one of our selected distributions as a proxy for the upstream release dates. We will somewhat abusively refer to the proxy as *upstream* in the rest of this paper. Even though our analyses focus on the period $[2015, 2020[$, we applied this procedure to all package versions between 2010 and 2020. This allowed us to approximate the upstream release date of package versions that were already available prior to 2015-01-01.

IV. Quantitative Analysis of Linux Distributions

RQ_1: How prevalent are outdated package versions in Linux distributions?

This research question aims to assess the prevalence of outdated package versions within Linux distributions. A version of a package in a distribution snapshot is *outdated* if a more recent version is present in another distribution on the same date. Fig. 1 shows the evolution of the proportion of outdated package versions in the considered Linux distributions.

The proportion of outdated package versions in Arch Linux is very low, oscillating between 5% and 17%. In contrast, 78% to 87% of the package versions in CentOS 7 are outdated. The proportion of outdated package versions in Debian Stable is around 50%. The proportion in Debian Unstable fluctuates

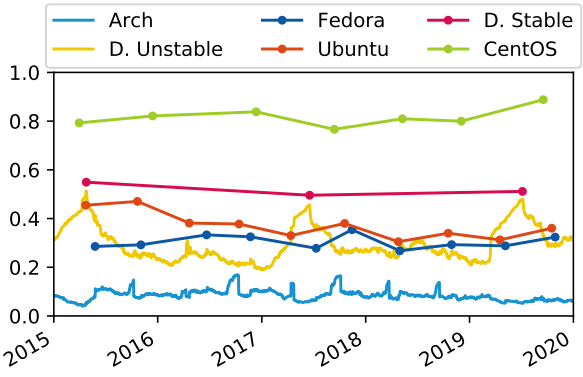

Fig. 1. Evolution of the proportion of outdated package versions.

wildly between 18% and 50%, peaking every 2 years. According to the Debian maintainers, this is due to the adopted release management policy: "*When a Testing release becomes 'frozen', Unstable tends to partially freeze as well. This is because developers are reluctant to upload radically new software to Unstable, in case the frozen software in Testing needs minor updates and to fix release critical bugs which keep Testing from becoming Stable*".[3] So, for a period of six months occurring roughly every year and a half, package versions in Debian Unstable become increasingly outdated, converging towards Debian Stable. Nevertheless, even at the points of the cycle when the fewest packages are outdated, Debian Unstable is more similar to Fedora and Ubuntu than to Arch Linux, which is unexpected for a distribution that serves as a development branch, where one would expect to see recent package versions being made available almost immediately in order to expedite the testing and validation process. In the case of Fedora, the proportion of outdated package versions hovers around 30%, peaking at 35% for Fedora 27. The proportion in Ubuntu has decreased over the past 5 years, starting at 44% in Ubuntu 15.04 and reaching 35% by Ubuntu 19.10. These observations suggest that not all distributions value keeping packages up-to-date to the same extent. This is mostly seen by contrasting Arch Linux with distributions that value concerns of stability over freshness, such as Debian Stable and CentOS, for which half to three-fourths of the packages are outdated.

Findings. The proportion of outdated package versions varies greatly between Linux distributions, from $\sim 10\%$ in Arch Linux to $\sim 80\%$ in CentOS. Despite being a development distribution, a significant proportion of packages in Debian Unstable are outdated.

RQ_2: How long have the versions been outdated?

RQ_1 established that some distributions use many outdated package versions. However, some versions may have been outdated for a very short time, while others have been outdated

[2]https://github.com/repology/libversion

[3]https://www.debian.org/doc/manuals/debian-faq/ftparchives.en.html#frozen

for many years. For instance, kscreen is outdated in Ubuntu 19.10 by a single day. Indeed, Ubuntu 19.10 was released on 2019-01-24 with kscreen 5.16.5, but version 5.17.0 was already available on 2019-10-23. At the other end of the spectrum, CentOS 7.6 shipped version 1.5 of package gzip even though version 1.6 had already been available for more than five years. It is therefore important to quantify the time during which distribution maintainers did not seize the opportunity to update. We thus measure for how long deployed package versions in distributions have been outdated, i.e. the time since a more recent upstream version has been available. To do so, we define the metric of ***update delay*** of a package p in a distribution as the time difference between the release date of the distribution containing p and the upstream release date of the first more recent version of p. If the distribution uses the latest version of p, the update delay is 0. Fig. 2 shows, in increments of 10%, the *proportion of packages in each distribution having at least a certain value of update delay* (in days) and in blue the evolution of the *mean update delay* (in days) per distribution over the observation period.

We see major differences between distributions. On average, packages in CentOS have been outdated by one order of magnitude longer than those in the other distributions. The opposite is observed for Arch Linux, which maintains a mean update delay of less than 52 days. We see diverging patterns between the Debian family of distributions (Debian and Ubuntu) and the Red Hat family (Fedora and CentOS). In the former case, the mean update delay grows relatively little over time, whereas in the latter case, it accrues rapidly over time. We observe a 87% increase in mean update delay between Fedora 22 (70 days) and Fedora 31 (131 days). Meanwhile, Ubuntu's mean update delay trends horizontally, allowing it to be lower than Fedora's by the time of Ubuntu 19.04. The mean update delay in CentOS almost triples (factor of 2.97) between CentOS 7.1 and CentOS 7.7, accruing a delay of 761 days over a period of 1,631 days. For Debian Stable, there is very little increase in mean update delay (16 days only) between release 8 and release 10, even though those releases are 1,533 days apart. The update delay of Debian Unstable fluctuates as a consequence of the "partial freeze" effect prior to Debian Stable releases, as observed in RQ_1.

In Arch Linux, the few packages that are sometimes outdated never remain so for long: their update delay rarely exceeds a few tens of days, indicating that maintainers quickly react and update packages to newly available versions. Its rolling release policy facilitates those quick reactions, as there is no need to wait for the next release to update packages. Few packages are outdated by more than 3 months in Fedora and Ubuntu: 10% in Fedora and some releases of Ubuntu, 20% in most releases of Ubuntu. 30% of Debian Stable packages use versions that have been outdated by 3 months or more. Debian Unstable oscillates between update delays that approach Debian Stable around the release of Debian Stable and update delays similar to or even lower than Fedora at other points of the Debian release cycle. CentOS stands in stark contrast with all other distributions, with always over half of its packages outdated by more than a year, and by more than 2 years by the time of CentOS 7.6. The discrepancy we observe between CentOS and the other distributions is partly explained by the fact that 41% of its packages have not been updated over the course of the observation period, despite being outdated from the first snapshot. In most distributions, this phenomenon only concerns $\leq 2\%$ of packages. For instance, version 0.15.1 of package aide was shipped in CentOS 7.7, despite the availability of a more recent version 0.16 on 2013-12-18, 2099 days prior (i.e. nearly 6 years)! In any distribution, amongst outdated package versions, at least 30% have an update delay of more than half a year and at least 20% of more than a year.

Findings. There is a large discrepancy in update delay between CentOS and other distributions: the mean update delay of the first considered CentOS release is more than thrice as high as the second distribution (Debian Stable) and ten times as high as the lowest distribution (Arch Linux), and the gap only widens with time. Most packages in most distributions have a relatively low update delay, below 3 months. This is not the case for CentOS, where half the packages have an update delay of more than a year. 20% of outdated packages could have been updated for more than a year in all considered distributions.

RQ_3: To which extent are deployed packages outdated?

The *update delay* provided information regarding the time since a package could have been updated to a prior version, but has not been. While informative, this paints an incomplete picture, as it does not measure the amplitude of the outdatedness. For instance, assume that version v_1 of package p, deployed in distribution d released on date x has an update delay of 2 days. Depending on when v_1 was released, p could be missing 2 days' worth of updates or years of updates. The update delay tells us that version v_2 was released on day $x-2$. If v_1 was released on day $x-3$, then d is only missing 2 days' worth of updates, but if v_1 was released on day $x-366$, then it is missing a full year's worth of updates.

We will quantify the amplitude of outdatedness of package versions in distributions compared to the upstream, by computing the difference between the release date of the most recent upstream version and the release date of the version deployed in a distribution. This corresponds to the ***time lag*** metric defined in the technical lag framework of Zerouali et al. [8]. For example, Fedora 31 (released on 2019-10-29) contains version 2.0.19 of package gob2. This version was released upstream on 2013-05-07 while version 2.0.20 was released on 2013-12-16. Thus, gob2 has a time lag of 223 days in Fedora 31. Fig. 3 shows, in increments of 10%, the *proportion of packages having this lower threshold of time lag* (in days) and in blue the evolution of the *mean time lag* (in days) per distribution over the observation period.

The mean time lag does not exceed 200 days for most distributions, with the exceptions of Debian Stable (rising to 205 days by Debian Stable 10), and CentOS that once again starts more than an order of magnitude higher than Arch Linux

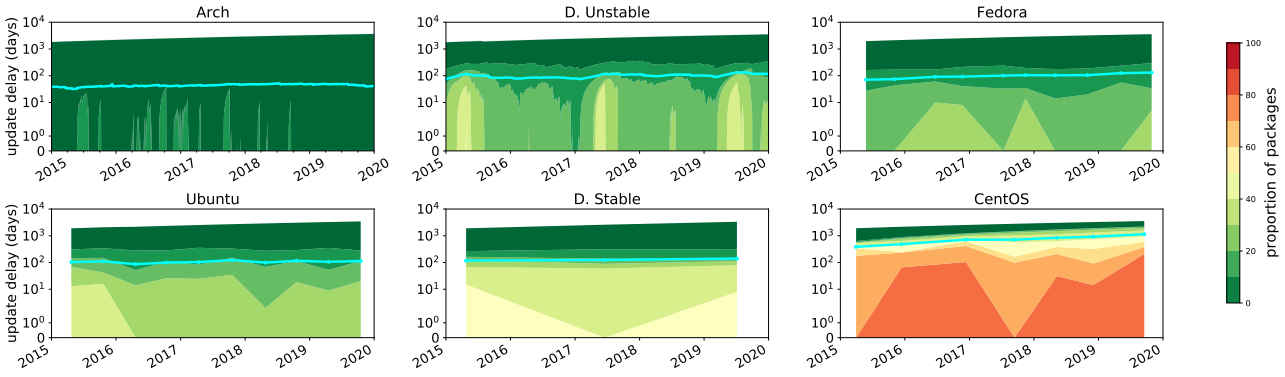

Fig. 2. Evolution (on logarithmic y-scale) of the *update delay* for each distribution. Mean update delay is shown in blue.

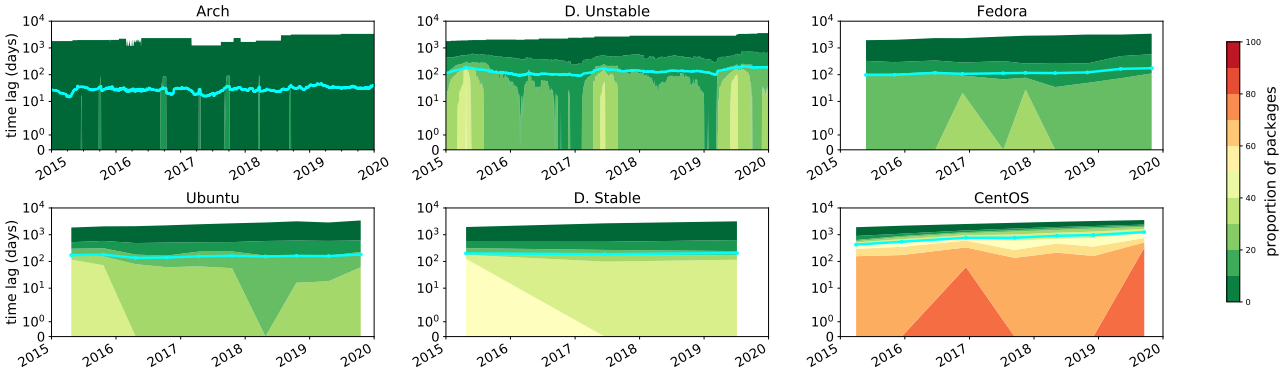

Fig. 3. Evolution (on logarithmic y-scale) of the *time lag* for each distribution. Mean update delay is shown in blue.

(427 versus 28 days) and climbs to over 3 years (1,252 days) by CentOS 7.7, more than 6 times higher than the time lag of Debian Stable. For the most part, most package versions in distributions are less than 3 months older than the latest available version: 50% in Debian Stable, 60% in Ubuntu and Debian Unstable, 70% in Fedora and 90% in Arch Linux. In Fedora, fewer than 20% of packages have a time lag of over 6 months. In Ubuntu, fewer than 30% do. Even in Debian Stable, more than 60% (Debian Stable 8) to 70% (Debian Stable 9 and 10) of packages are using versions less than 6 months old. As prior, Debian Unstable fluctuates according to the Debian release cycle. CentOS is again the exception: half of its packages are outdated by more than a year and a further 10% by more than 6 months in CentOS 7.1. By the time of CentOS 7.7, more than 70% are outdated by more than a year and more than 30% by more than 5 years! Considering only the outdated packages in any distribution, at least half of them have a time lag of more than six months, and 30% of $>$ 1 year.

Findings. Distributions have an average time lag of roughly half a year, with the exceptions of Arch Linux (hovering around a month) and CentOS (from 1 to 3 years), at opposite ends of the spectrum. The majority of package versions in 5 out of 6 distributions are less than 3 months older than the latest available version. At least 30% of the outdated packages are missing more than a year of updates.

The metrics of update delay and time lag are complementary, capturing different facets of outdatedness. Two packages can therefore have the same update delay but a very different time lag. For example, packages exempi and enchant in Debian Stable 9 both have similar update delays (133 and 127 days, respectively), but a very different time lag (12 days and 2454 days, respectively). Conversely, packages xdg-user-dirs and libmpc in CentOS 7.5 both have a time lag of 1722 days, but a very different update delay (246 days and 1564 days, respectively). A high time lag is understandable if the update delay is small, as the distribution maintainers have not had much time to incorporate the new version(s). A high update delay coupled with a low time lag can indicate that the distribution maintainers deliberately skipped a version that introduced an undesired change. Whereas if both metrics are high, the distribution might be lacking important bug fixes or features. At the distribution level, the observations we made are consistent across both metrics.

Lessons learned. At package level, *update delay* and *time lag* capture different facets of package outdatedness. Nevertheless, at the level of Linux distributions, both metrics are consistent, since the metrics only show important differences for a minority of packages.

V. Discussion

A. Importance and impact of package freshness

A survey of Linux users [5] revealed that 75% of them value package freshness and that this is more prevalent for users of

cutting edge distributions like Arch Linux and Fedora than users of reputedly stable distributions like Debian Stable and CentOS. Installing outdated packages exposes their users to the risk of known security vulnerabilities (unless the security patches have been backported) and users miss out on both bug fixes and new features. Using an outdated version of package p_1 may also prevent the adoption of new versions of package p_2 that depend on a fresher version of p_1. The aforementioned survey also revealed that users are inclined to update packages through the official repositories whenever possible. This can be explained by the fact that using official repositories comes with the benefit of knowing that the packages have been tested for bugs, co-installability with other packages, dependency requirements and security vulnerabilities and that the user will be notified when future versions are deployed. Yet, packages are not always fresh in the official repositories of the distributions. Even in distributions that prioritise freshness, one can find outdated packages. For instance, Arch Linux shipped version 4.4.2 of package findutils until 2016-02-27, despite more recent versions being available since at least 2011-11-08 (version 4.5.9, present in Fedora 16). This can have several causes, such as the maintainers being reluctant to update to a version that causes excessive breaking changes or incompatibility of the package with other components of the distribution. Whenever a package is not fresh in (or absent from) the official repositories, other means may be used to install the required package version, such as using community repositories (e.g. PPAs for Ubuntu, RPM Fusion for Fedora), third-party package managers (e.g. Flatpak, Snappy) or precompiled binaries. The survey confirmed that, for instance, 39% of users resort to binaries to update proprietary software and more than a third use community repositories to install both open-source and proprietary software. In using these means, users do not profit from the benefits of official repositories. Some, such as installing binaries directly, even carry a risk that the package is malware. It is therefore important that packages in the official repositories be fresh.

B. Ranking distributions in order of freshness

Given the above, we propose to rank distributions in terms of freshness, thereby informing user choice. To do so, we look at the packages found in a snapshot of each distribution and rank distributions based on the freshness of these packages: the distribution with the highest freshness for a given package receives a position of 1, the one with the lowest freshness a position of 6). Ties are handled using standard competition ranking.[4] We use snapshots of the latest release of point-release distributions. The snapshots used for Arch Linux and Debian Unstable are those on the date of the last point-release snapshot (Fedora). As a result, the snapshots used for most distributions are within two weeks of each other, except CentOS (1 month older) and Debian Stable (3 months older).

We computed the freshness rankings using both update delay and time lag. Since we obtained similar results for both metrics, we only report the time lag ranking. Fig. 4 shows, for each distribution, the proportion of packages with regards to their time lag rank. The numbers shown are cumulative, so a column n shows the proportion of packages ranked from position 1 to n. For example, the 76% value for Fedora in the rank 2 column means that 76% of its packages have a rank 2 or lower. Since 71% of its packages have a rank of 1, that means 5% have a rank of 2. Five out of six distributions place first more often than not. This is explained by the fact that from 49% (Debian Stable) to 93% (Arch Linux) of packages in the selected snapshot of those distributions are up-to-date, therefore it is expected that many of these packages will tie for first (time lag of 0). Arch Linux emerges as the clear winner, its packages very rarely being ranked anything but first. Then comes a peloton of 3 distributions: Fedora, Debian Unstable and Ubuntu, ranked first roughly two-thirds of the time. Debian Stable stands slightly back; its packages are ranked first more than half the time (53%) but lagging behind the packages in the four prior distributions a fourth of the time (25%). CentOS is far behind, its packages being ranked sixth 71% of the time, which is consistent with the results obtained in prior analyses.

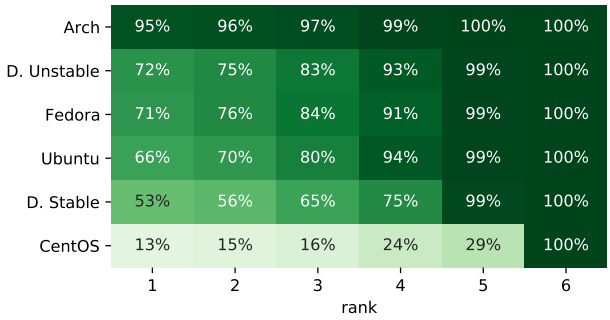

Fig. 4. Distribution of time lag rankings for packages in Linux distributions.

To verify whether there is a significant statistical difference between distributions, we compared the distribution of time lag values between each pair of Linux distributions with a two-sided Mann-Whitney U test [16]. For most pairs of Linux distributions, we could reject the hypothesis that the statistical distributions of lag values are identical with statistical significance ($p < 0.01$ after Holm-Bonferroni correction [17]), but could not do so for Ubuntu and Fedora, Ubuntu and Debian Unstable and Fedora and Debian Unstable. We measured the effect size of the difference in time lag between distributions with Cliff's delta d [18]. Fig. 5 reports the results in a Hasse diagram. An edge from a distribution D_1 to D_2 indicates that D_1 has fresher packages than D_2, the reported value is the effect size. All effect sizes are *small* (and *medium* for Arch $\rightarrow$ Ubuntu) following the interpretation of Vargha and Delaney [19]. An order of the relative package freshness of distributions can be inferred, with Arch Linux being the most fresh, followed by the trio Fedora, Ubuntu and Debian Unstable, then Debian Stable, and finally CentOS. Although Debian Unstable comes second with regards to overall package

[4]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rankdata.html

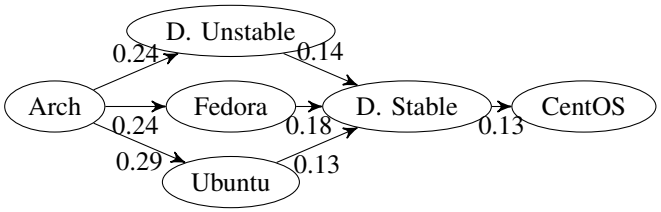

Fig. 5. Order of Linux distributions based on statistical comparison of the time lag of their packages

freshness, it fluctuates appreciably in accordance with the release cycle of Debian Stable.

Recommendation. Users that place primordial importance on package freshness should default to Arch Linux. As a compromise between package freshness and other factors (such as ease of use or total number of packages), we recommend using Fedora or Ubuntu.

C. Comparing Linux user perceptions with quantitative data

According to our survey [5], Arch Linux users perceived it took on the order of days for upstream versions of packages to be released in the Arch Linux repositories. We quantitatively confirm that these user perceptions match reality for the majority of packages, as the time lag of 90% of packages rarely exceeds ten days. Still, a handful of packages have a significant time lag. For instance, version 1.2.3 of package cdrdao (available since 2010-05-25) is present in Arch Linux until 2018-11-13, even though version 1.2.4 had been available since 2018-07-22, resulting in a time lag of 2980 days (over 8 years). Fedora and Ubuntu users deemed that it took on the order of weeks. This perception is mostly accurate, as 60% of packages have a time lag of less than a month in all releases of Fedora and all but 3 releases of Ubuntu (15.04, 15.10, 17.10). Debian Stable and CentOS users considered that it took on the order of months for upstream package versions to reach the downstream official repositories of their distribution. For Debian Stable, that perception is essentially correct: only 30% to 40% of packages have a time lag higher than six months. By contrast, in the case of CentOS, it is more appropriate to talk in years as half of its packages have been outdated by over a year in all considered releases.

Lesson learned. The user perception of the time it takes for packages to be deployed within their distribution roughly approximates reality, with the exception of CentOS users who largely underestimate this time.

VI. Threats to validity

We discuss the threats to the validity of our findings, following the structure established in [20].

Construct validity concerns the appropriateness of using the findings of the experiments undertaken in a study to make inferences, i.e. do the experiments measure what they are supposed to. The principal construct validity threats are related to the use of a proxy (see III) to approximate the upstream release date of package versions. We did so due to the lack of complete and centralised sources on package version history. As a result, the release date we consider does not reflect the date when this version was released by the package authors, but the date when this version was first witnessed in one of our distributions. Versions which do not appear in any of the six distributions are not part of the proxy, and those that do appear are assigned a release date that is likely posterior to their actual release date. As such, we potentially underestimated the true update delay, time lag and proportion of outdated packages in the distributions. As we are principally concerned with comparing the freshness of Linux distributions, underestimating these values is not really an issue, as it applies to all distributions studied.

Additionally, we used snapshots of point-release distributions at the time a new release is made. This does not consider the evolution of the distribution, such as updates to packages in the repository of a distribution made in between two releases. Therefore, for point-release distributions, we only present interpolations of the behaviour between snapshots.

Internal validity concerns the impact of the choices made in carrying out a study on its results. The use of a proxy can make lag appear more sudden than it really is: a new release of one distribution will make a set of packages of other distributions appear suddenly out of date, when in fact those packages did not all release new versions on the same date. This effect corrects itself whenever those packages are updated. This effect is especially apparent in distributions with a rolling release policy, for which we chose to use daily snapshots. For instance, in Fig. 1 we observed that the release of Ubuntu 17.10 on 2017-04-13 caused some packages in Arch Linux to suddenly appear outdated, but this small spike was short-lived as those packages were updated a few days later in Arch Linux. For that reason, we restrained from making observations on local events happening over a few days.

Conclusion validity concerns the reasonableness of the conclusions derived from a study. There exist two categories of threats to conclusion validity: (a) not detecting a relationship that exist and (b) detecting relationships where there are none. In both cases, the statistical power of the analyses carried out plays an important role. A potential threat to conclusion validity comes from the relatively small size of our final dataset of 890 packages compared to the total number of packages in some distributions. However, using Cochran's sample size formula, adjusted for a finite population as per [21], our sample size of 890 is representative for even the biggest set of packages in our distributions ($\sim 62,000$, in latter snapshots of Debian Stable and Debian Unstable) within a margin of error of 4% at a confidence interval of 95%.

External validity concerns the generalisability of the conclusions of a study to a larger scope. A threat to external validity lies in selecting only packages that are present in all distributions. This biases the selection in favour of packages that are widespread and generally established in the community over more novel packages. As such, our findings are not generalisable to all other packages. One could measure the

freshness of all packages in all distributions, and compare the freshness of the distributions on this basis. However, such a comparison would be unfair if the set of considered packages is not the same for all distributions.

VII. Conclusion

Linux distributions rely on external packages to provide their users functionalities that extend beyond those of the Linux kernel. These packages are gathered in the distributions' repositories. Yet, the package versions found in these repositories are not always the latest available ones. A survey of Linux users [5] reported that they value package freshness in the official repositories of the distributions they use. In order to allow users to make an informed choice, we therefore assessed the *freshness* of packages in Linux distributions using the complementary metrics of update delay and time lag to measure how up-to-date packages deployed in Linux distributions are compared to the available upstream releases.

We examined the proportion of outdated packages in six Linux distributions, finding a large discrepancy between the most up-to-date distribution (Arch Linux) and the most outdated one (CentOS), in terms of the number of outdated packages, update delay and time lag. As such, users of distributions such as CentOS and, to a lesser extent, Debian Stable benefit from new features and bug fixes after other Linux users and might be exposed to known vulnerabilities for a longer period of time. On the other hand, packages being deployed after more extensive testing should better shield those users against stability issues. From our analyses, Arch Linux emerges as by far the most up-to-date distribution, followed by a trio of distributions (Fedora, Ubuntu and Debian Unstable), then by Debian Stable. CentOS is far behind the others. Most packages in Arch Linux, Ubuntu and Fedora are up-to-date, and those that are not are rarely outdated by more than a single version. While Debian Unstable is sometimes as fresh as Fedora and Ubuntu, this highly depends on the Debian release management cycle. We therefore would recommend to users who value having fresh packages to choose between Arch Linux, Fedora and Ubuntu. Surveyed Linux users appear to have an accurate idea of the time it takes to deploy package updates within their distribution, with the exception of CentOS users who underestimated that time.

Future work will focus on examining the trade-offs between freshness, security and stability, and ranking distributions according to an aggregate of these criteria. We also aim to explore the impact of third-party package managers, such as Flatpak, Snappy and AppImage on package freshness: the existence of those package managers grants Linux users another avenue to obtain packages, and therefore potentially mitigate the potentially poor freshness of their chosen distribution. We will examine whether the packages in those package managers are fresh and numerous enough to accomplish this objective.

Acknowledgment

This research is supported by the Fonds de la Recherche Scientifique – FNRS under Grants number O.0157.18F-RG43 (Excellence of Science project SECO-ASSIST) and T.0017.18.

References

[1] Godfrey and Qiang Tu, "Evolution in open source software: a case study," in *International Conference on Software Maintenance*, Oct 2000, pp. 131–142.

[2] G. Robles, J. J. Amor, J. M. Gonzalez-Barahona, and I. Herraiz, "Evolution and growth in large libre software projects," in *International Workshop on Principles of Software Evolution*. IEEE, 2005, pp. 165–174.

[3] J. Vouillon and R. Di Cosmo, "On software component co-installability," in *Joint European Software Engineering Conference / Foundations of Software Engineering*, 2011, pp. 256–266.

[4] J. Rutkowska and R. Wojtczuk, "Qubes OS architecture," Invisible Things Lab, Tech. Rep. 54, 2010.

[5] D. Legay, A. Decan, and T. Mens, "On package freshness in linux distributions," in *International Conference on Software Maintenance and Evolution*. IEEE Computer Society, oct 2020, pp. 682–686.

[6] J. Cox, E. Bouwers, M. van Eekelen, and J. Visser, "Measuring dependency freshness in software systems," in *International Conference on Software Engineering*. IEEE Press, 2015, pp. 109–118.

[7] J. M. Gonzalez-Barahona, P. Sherwood, G. Robles, and D. Izquierdo, "Technical lag in software compilations: Measuring how outdated a software deployment is," in *IFIP International Conference on Open Source Systems*. Springer, 2017, pp. 182–192.

[8] A. Zerouali, T. Mens, J. Gonzalez-Barahona, A. Decan, E. Constantinou, and G. Robles, "A formal framework for measuring technical lag in component repositories – and its application to npm," *Journal of Software: Evolution and Process*, vol. 31, no. 8, 2019.

[9] J. Vouillon and R. Di Cosmo, "Broken sets in software repository evolution," in *International Conference on Software Engineering*, 2013, pp. 412–421.

[10] M. Claes, T. Mens, R. D. Cosmo, and J. Vouillon, "A historical analysis of Debian package incompatibilities," in *Working Conference on Mining Software Repositories*, 2015, pp. 212–223.

[11] C. Artho, K. Suzaki, R. Di Cosmo, R. Treinen, and S. Zacchiroli, "Why do software packages conflict?" in *Working Conference on Mining Software Repositories*, 2012, pp. 141–150.

[12] J. A. Galindo, D. Benavides, and S. Segura, "Debian packages repositories as software product line models – towards automated analysis," in *ACoTA*, 2010, pp. 29–34.

[13] R. Nguyen and R. Holt, "Life and death of software packages: an evolutionary study of Debian," in *Conference of the Center for Advanced Studies on Collaborative Research*, 2012, pp. 192–204.

[14] J. M. Gonzalez-Barahona, G. Robles, M. Michlmayr, J. J. Amor, and D. M. German, "Macro-level software evolution: a case study of a large software compilation," *Empirical Software Engineering*, vol. 14, no. 3, pp. 262–285, 2009.

[15] S. Shawcroft, "Open source watershed: Studying the relationship between Linux package and distribution releases," Bachelor Thesis, http://oswatershed.org, June 2009.

[16] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 03 1947.

[17] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.

[18] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[19] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[20] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[21] G. D. Israel, "Determining sample size," University of Florida, Institute of Food and Agricultural Sciences, Tech. Rep. PEOD6, 1992.