Journal of Computational Interdisciplinary Sciences (2009) 1(2): 127-133
© 2009 Pan-American Association of Computational Interdisciplinary Sciences
ISSN 1983-8409
http://epacis.org

Analysis of the package dependency on Debian GNU/Linux

O. Felicio de Sousa, M.A. de Menezes and Thadeu J.P. Penna

Manuscript received on January 12, 2009 / accepted on March 10, 2009

ABSTRACT

We build the network of more than 18,000 software packages of Debian GNU/Linux operating system, for its current three releases, along with their dependence interactions. We measured the degree connectivity distribution, modularity and detect some communities. The scale-free nature of the dependency network and the results for the betweenness centrality can confirm the stability of the system to random bugs. Also the communities we have found can be used for determining the packages that could be mantained by teams rather than developers.

Keywords: open source projects, communities, complex networks.

Correspondence to: Thadeu J.P. Penna
Instituto de Física, Universidade Federal Fluminense, Av. Litorânea s/n, 24210-340 Niterói, RJ, Brazil and Instituto Nacional de Ciência e Tecnologia – Sistemas Complexos – INCT-SC, Brazil.
E-mails: orahcio@if.uff.br / marcio@if.uff.br / tjpp@if.uff.br

1 INTRODUCTION

In the last years an increasing number of systems have been described as networks (*i.e.*, a set of nodes connected between them by links) and represented as graphs [1]. In this work we compiled the network of software packages of Debian GNU/Linux operating system along with their dependence interactions in order to better understand the stability of such system. The Debian GNU/Linux operating system is one of the most popular GNU/Linux distributions, not only among end users but also as a basis for many other distributions. It is also one of the largest software compilations. Its stable version has almost 18,000 packages, counting more of 250,000,000 lines of source code.

The Debian GNU/Linux operating system is maintained by the Debian Project, that is a worldwide group of volunteers: the Debian developers, with almost 1,500 members today. The system which includes the Linux operating system kernel, and thousands of prepackaged applications. With the purpose of being the Universal Operating System, Debian GNU/Linux is available for several architectures, including Intel x86, amd64, PowerPC, Alpha and SPARC. Although the kernel of the Debian GNU/Linux software distribution is the Linux kernel, there are other versions that are based on different kernels like Hurd and FreeBSD's.

Because of the free spirit of Debian, it is very easy to get informations about the project and its evolution. There are other works on the data from the Debian Project, and we would like to cite two of them here to stress the differences between our work and the previous ones. Maillart et al. [5] studied the time evolution of the links between the packages in order to demonstrate empirically the conditions under which a distribution obeying Zipf's law occurs. They focused on how the links to certain package changes with time. Because the fluctuations in the number of links also grows with the number of links, there is a finite probability of a highly connected package gradually disappear. Here we study a different problem that is how a bug in a given package can affect the whole distribution. Another application of the Debian packages network is the work by Fortuna and Melian [3]. There the authors mimics a gene regulatory network using the relations of dependences and conflicts of the Debian packages. They found that complex networks allow a larger active network size than random ones, by studying the maximum number of packages that can be installed at once.

1.1 Debian releases

In addition to the stable branch (named as "etch", today), the Debian Project include two other development releases: testing and unstable. After a freeze stage in which only packages fixing bugs were included in the testing release, development started in this release after the stable has been delivered. The unstable release (which is always known as sid) is used by developers to upload new package versions (which includes new versions of the package software), and test them.

The codename of the new stable release will be lenny. In order to guarantee that the testing release is always in a releasable state, changes to packages need to be uploaded to unstable. If no important bugs are filed against the new packages nor to the packages they depend on for ten days, the packages are automatically included in testing and considered for inclusing in the next stable release.

The Debian Free Software Guidelines (DFSG) defines what Debian understand as free software. The Open Source definition is actually derived from DFSG (see http://www.us.debian.org/social_contract). Software that can be freely distributable and also modifiable are in agreement with the DFSG. The full archive of Debian packages is composed of three categories: main, contrib and non-free. The main category is made up of packages that comply with the DFSG and do not require any non-DFSG packages. On the other hand, non-free is the category made up of packages not compliant with the DFSG (but can be distributed by Debian) or are encumbered by patents or other legal issues that make their distribution problematic. Finally, contrib is the category made up of packages that comply with the DFSG but require a non-DFSG package. In this work, we will be dealing with the full archive, including the three categories.

1.2 Packages and their dependencies

The packages in Debian can be of two kind: source packages and binary packages. Binary packages provide the programs compiled for the set of computer architectures Debian supports. Frequently, more than one binary packages can be automatically built from only one source package. For example, a source package of a program can be divided in a binary package for the program itself, a binary package for its libraries (if they can be reused by other programs) and a binary package for its documentation. This division is important to understand the dependencies network. So, in order to install a given application, it will be neccessary to download the packages with contains the libraries that it depends on. Debian APT (Advanced Packaging Tool) is used to make the installation of the packages an easy task. The packaging system uses a private database to keep track of which packages are installed, which are not installed and which are available for installation. The apt-get program uses this database to find out how to install packages requested by the user and to find out which additional packages are needed in order for a selected package to work properly.

A source package contains the original source (got from a ftp site, etc) of a piece of software. It is called the "upstream" source. The Debian developer patches upstream sources if needed, and creates a directory debian with all the Debian configuration files (including data needed to build the binary package). Then, the source package is built, usually consisting of three files: the upstream sources (a tar.gz file), the patches to get the Debian source directory (a diff.gz file, that can include both patches to upstream sources and the debian directory), and a description file. In most cases, one developer is responsible for each Debian package. Some particular large and crucial packages are kept by teams (see http://www.debian.org/devel/people).

2 RESULTS

The modular structure of Linux distributions is one of its great features, which descentralizes the work done by the operating system when performing distinct tasks like controlling a TCP port, reading a CD-ROM or compiling a C code. For such, there is a hierarchy of softwares, or "packages", which operate in a dependent way. We reconstruct such dependency network for a Debian distribution, with information from http://ftp.br.debian.org/debian/dists/etch/main/binary-i386/Packages.bz2 which displays, for each package, a list of packages it depends on and must necessarily be installed by using APT. We work with main, contrib and non-free sections and etch, lenny and sid releases. We reconstruct the package dependency graph $\mathcal{G}(\{V\}, \{E\})$ assigning a vertex $i \in \{V\}$ to each package of the entire Linux distribution and a directed edge $e = (i, j) \in \{E\}$ between nodes i and j if j depends on i (or, equivalently, if for j to be installed i needs to be installed first) and find $N = |V| = 18450$ nodes and $M = |E| = 34979$ edges for the Etchy, 23802 nodes and 42395 edges for the Lenny and 23963 nodes and 44166 edges for the Sid distribution.

For each node i we calculate how many packages it depends on $k_{in}(i)$ and how many depend on it, $k_{out}(i)$. We found broad (power-law) scaling for the out-degree distribution, $P_{out}(k_{out}) \sim k_{out}^{\gamma}$, with $\gamma \simeq 2$, indicating that packages play very different roles in the distribution, with few essential packages and many specific ones (Fig. 1). On the other hand, the in-degree distribution $P(k_{in})$ is much narrower (see inset of Fig. 1), and can be fit either by an exponential $Ae^{-k_{in}/\xi}$ with $A \simeq 10^3$ and $\xi \simeq 2.0$ or a power-law with $\gamma \simeq 4.0$.

We further proceed to analyse higher-order patterns, or motifs, in the Debian dependency network. We follow [2] and de-

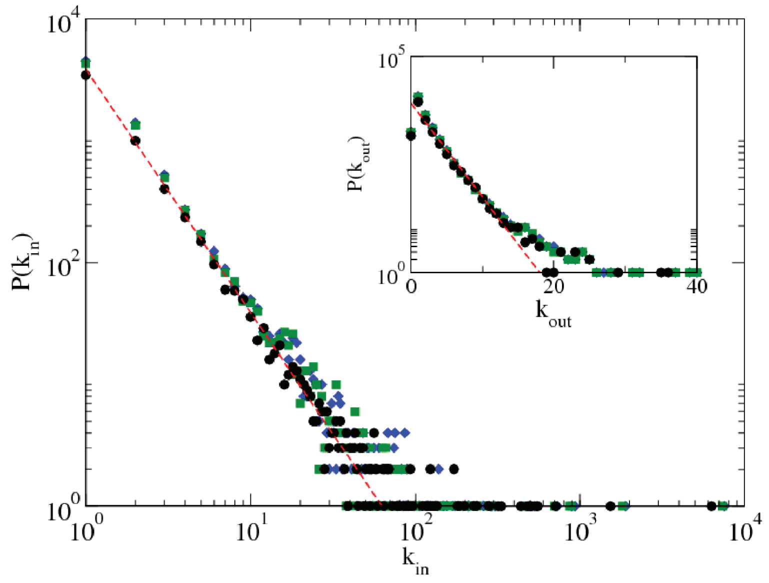

Figure 1 – In-degree and out-degree connectivity for the Debian Etch dependency network. The out degree measures how many packages depend on a given one and displays a broad distribution $P(k_{out}) \sim \gamma$ with $\gamma \simeq 2.0$. Inset: The in-degree, which measures how fragile or dependent is a given package, displays a much narrower (exponential-like) distribution.

fine 4 different triangles: Let $\mathcal{A}$ be the connectivity matrix with $a_{ij} = 1$ if there is a link between nodes i and j, and 0 otherwise, the density of *in*, *out*, *middleman* and *cycle* triangles passing through node i can be written as

$$c_i^{in} = \frac{\sum_{j,k} a_{ji} a_{jk} a_{ki}}{M_a}, \quad (1)$$

$$c_i^{out} = \frac{\sum_{j,k} a_{ij} a_{jk} a_{ik}}{M_b}, \quad (2)$$

$$c_i^{mid} = \frac{\sum_{j,k} a_{ij} a_{kj} a_{ki}}{M_c}, \quad (3)$$

$$c_i^{cyc} = \frac{\sum_{j,k} a_{ij} a_{jk} a_{ki}}{M_c}, \quad (4)$$

with the normalizations

$$M_a = \sum_{j,k} a_{ji} a_{ki}, \qquad M_b = \sum_{j,k} a_{ij} a_{ik} \quad \text{and} \quad M_c = \sum_{j,k} a_{ij} a_{ki} \quad (5)$$

such that each node has a clustering vector

$$\vec{c}_i = \left(c_i^{in}, c_i^{out}, c_i^{mid}, c_i^{cyc}\right), \quad (6)$$

and one can write the components of the network clustering vector $\vec{C}$ by averaging over all nodes of the network

$$C^k = \frac{1}{N} \sum_i c_i^k \quad (7)$$

On Table 1, we show the value of such clustering coefficients averaged over all nodes of the graph of three Debian releases (Etch, Lenny and Sid) and compare it to the expected results for its undirected counterpart and for random Erdös-Rényi graphs $c_{ER} = M/N(N-1)$. The systematically higher-than-random values for some type of clustering indicates some packages cluster into groups or regions. The cycle-type of triangle is statistically irrelevant, while all the others appear as a result of the modular structure of the Linux operating system.

Insted of measuring the damage associated with a given package, one can also study the fragility of a given package or how is a package affected if another, randomly chosen, package has a bug. Let us recall the definition of betweenness centrality of a given node, $b_{xy}(i)$, as the normalized number of shortest paths, or geodesics, between nodes x and y that pass through node i. Averaging over all L paths of the network we obtain

$$B(k) = \frac{1}{L} \sum_{x \in \{V\}} \sum_{y \in \{V\}} \frac{b_{xy}(k)}{\sum_k b_{xy}(k)}. \quad (8)$$

We plot in Figure 2 the betweenness distribution $P(B)$ for the Linux dependency network. With such measure one can infer where are the less fragile nodes, which have small betweenness B, and the critical ones, nodes with high betweenness that are at bottlenecks: they are central to others but at the same time very much affected by occasional bugs in other packages. For the Linux distributions studied we find reasonable balance between centrality of hubs and damage spreading.

As modularity is a key idea in Linux development, we try to define such modules from the connectivity patterns of the dependency network. We use a Potts-like model to infer functional modules, where the strength of the interactions J_{ij} between nodes i and j reflect the deviations from a null-model expectation for their connection [6, 4]

$$J_{ij} = a_{ij} - \frac{k_i^{in} k_j^{out}}{M} \quad (9)$$

where $k_i^{in} = \sum_j a_{ji}$ and $k_i^{out} = \sum_j a_{ij}$. Assigning a spin variable $s_j = 1 \dots Q$ to each package we can translate the problem of finding a modular subdivision of the network to the one of finding the ground state of the spin Hamiltonian

$$H = -\frac{1}{M} \sum_{ij} \left[a_{ij} - \frac{k_i^{in} k_j^{out}}{M} \right] \delta_{\sigma_i \sigma_j} \quad (10)$$

We solved the problem with Simulated Annealing and present the results in Figure 3. A tipical community (xfonts) is displayed in Figure 4. We chose a small community for clarity.

Table 1 – Comparison between clustering coefficients, for each release and an Erdös-Rényi equivalent network. The clusters of *out* triangles are more likely. Cyclic triangles corresponds to non-installable packages and are bugs of the release.

	C^{cyc}	C^{mid}	C^{in}	C^{out}	C	c_{ER}
Etch	$2.1291 \cdot 10^{-4}$	0.0365	0.0133	0.0862	0.0979	$2.4893 \cdot 10^{-4}$
Lenny	$1.4645 \cdot 10^{-4}$	0.0454	0.0164	0.1029	0.1177	$1.9113 \cdot 10^{-4}$
Sid	$1.9964 \cdot 10^{-4}$	0.0454	0.0165	0.1019	0.1164	$1.8276 \cdot 10^{-4}$

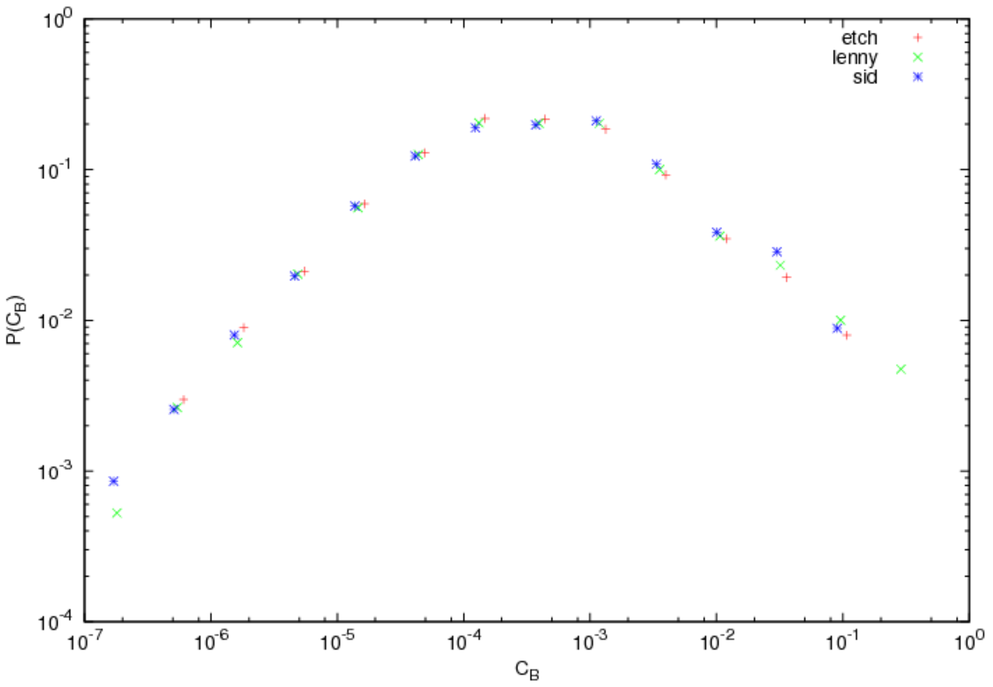

Figure 2 – Betweenness centrality for the debian releases, etch ($+$), lenny ($\times$) and sid ($\bigstar$).

It is interesting to comment on this last result: the largest community corresponds to the applications that depends on X, *i.e.*, applications that run on the graphical mode. Perl and Python follows libc6 that is the package for the applications written in C (more popular in the Unix world). Apache, the webserver, has its own community due the large number of modules that are optional and then packaged individually. The community debconf corresponds to the packages that are used to maintain the packaging system (APT). According this method of community detection there are a lot of other communities of but with a very small size.

3 CONCLUSIONS

In this work we compiled the network of software packages of Debian GNU/Linux operating system along with their dependence interactions in order to better understand the stability of such system. We succeded in finding a very small number of packaging bugs in the stable releases, corresponding to non-installable packages. The scale-free nature of the packages dependencies and the betweenness centrality have confirmed the robustness and stability of the system, considering random bugs but vunerable to directed ones. Our study of community could help the Debian Project by, for example, by suggesting a criteria for creation of packaging teams.

REFERENCES

[1] ALBERT R & BARABASI AL. 2002. Statistical mechanics of complex networks. Reviews of Modern Physics, 74(1).

[2] FAGIOLO G. 2007. Clustering in complex directed networks. Physical Review E (Statistical, Nonlinear and Soft Matter Physics), 76(2).

[3] FORTUNA MA and MELIÁN CJ. 2007. Do scale-free regulatory networks allow more expression than random ones? Journal of Theoretical Biology, 247(2): 331–336, July.

[4] LEICHT EA and NEWMAN MEJ. 2008. Community structure in directed networks. Physical Review Letters, 100(11).

[5] MAILLART T, SORNETTE D, SPAETH S & von KROGH G. 2008. Empirical tests of zipf's law mechanism in open source linux distribution. Physical Review Letters, 101(21): 218701.

[6] REICHARDT J & BORNHOLDT S. 2004. Detecting fuzzy community structures in complex networks with a potts model. Physical Review Letters, 93(21).

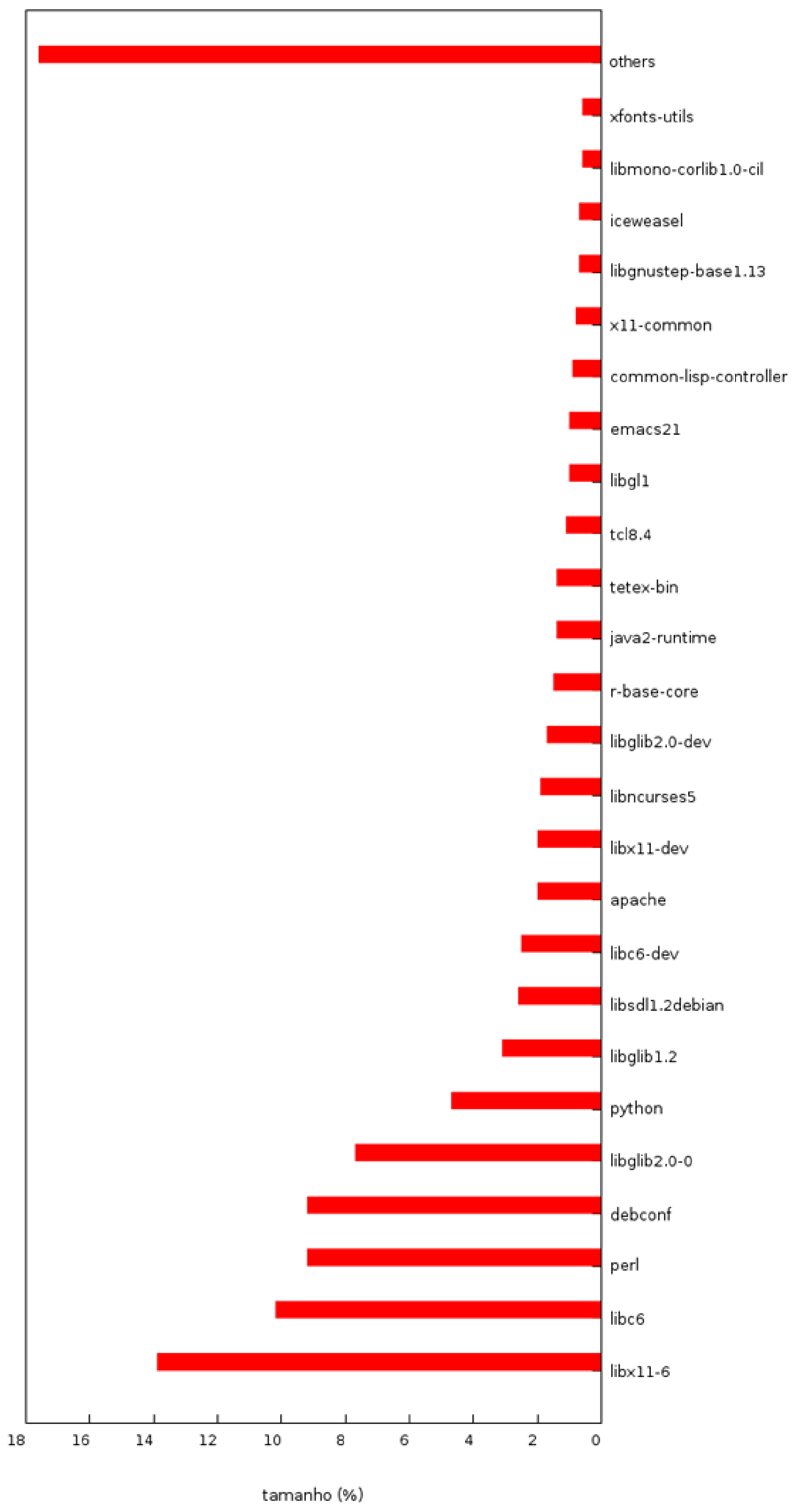

Figure 3 – Communities found through simulated annealing.

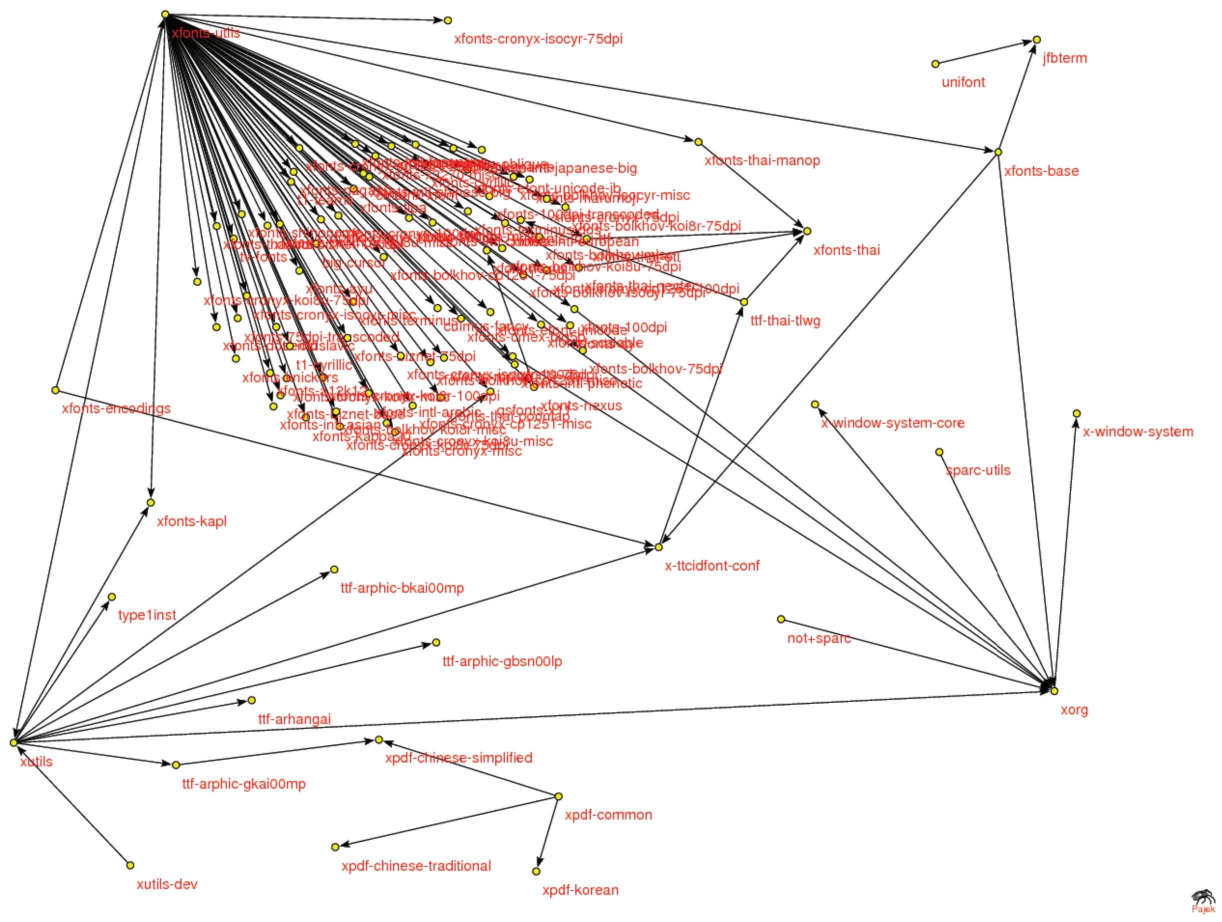

Figure 4 – Community driven by xfonts package that is a dependency for all fonts and some packages. It is not a big community but it is easier to find the connections between the packages.