The software package dependency networks of some Linux distributions

Árpád Horváth
Óbuda University, Alba Regia University Centre
H-8000 Székesfehérvár, Budai út 45., Hungary
Email: horvath.arpad@arek.uni-obuda.hu

***Abstract*—We have developed a cxnet software in Python to create, store and investigate the network of the software packages of Linux distributions that uses deb packages.**

We have investigated the software package network of three Linux distributions: Ubuntu, Debian and Arch Linux. For Ubuntu we have found, that between two versions the new incoming edges for the packages is nearly proportional with their in-degree.

I. Introduction

The GNU/Linux operating system has a lot of distributions, that have their own software packages. In these packages can be *binary files*, that means the programs in a distribution can be precompiled for one or more processor architecture. Other packages can include *source files* with some rules for the compilation. One package can depend on other packages, that means without them it is not able to function properly, so the packages compose a directed network. We defined the direction of the edges to direct from the dependent package to the package it depends on.

One of the software package formats is the *deb* package format, a binary format developed by the *Debian distribution* team [1] and used many other distributions, including the popular *Ubuntu*. The packages of the Debian are precompiled for many processor architecture (Intel, AMD and so on) and can be reached via repositories on the World Wide Web. These dependencies are stored in other files than software packages, so we need to download only these files to create the network of the packages. These distributions use the *APT (Advanced Packaging Tool)* to update these files. We have developed a Python program called *cxnet*, that can get the software package network of Debian and Ubuntu, and other distributions using *deb* packet format. This program uses the *apt* package of the Python programming language to get the network [2]. In our program the edges are not restricted to dependencies. It uses three types of connections: dependencies, recommendations and provisions, and two types of packages: real and virtual. These types will be discussed below.

For Arch Linux the pacgraph Python package can collect the packages and its dependencies from the official repository.

The `cxnet` package can create, store and analyse the package dependency network of the GNU/Linux operational system. We use it in the higher education to teach complex networks. It is based on the three packages:

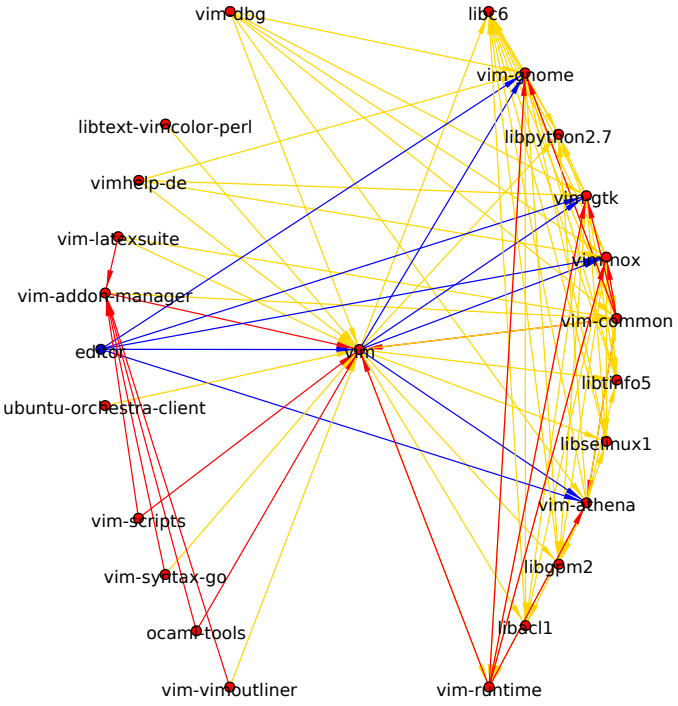

Figure 1. A part of the package dependency network of Ubuntu: the surroundings of the package of the Vim text editor. This plot includes the package with the highest degree, the libc6 package, the standard library of the C programming language.

1) igraph complex network analyser, that provides an efficient data structure and useful functions to investigate networks
2) the apt Python package, that can create the dependency network,
3) the pylab (and numpy) package, that can plot diagrams similarly as in MATLAB.

The installation, the tutorial and the library reference of the package can be reached at [3].

The `cxnet` can store the network properties into compressed gml and graphml file format, read them from these formats and analyse them using the igraph complex network analysing tool.

II. *deb* package based distributions

We have investigated the software package network of Ubuntu earlier [4], [5]. In that articles we wrote about the

properties of the package dependency network of one Ubuntu version. In this one we compare more versions of Ubuntu and Debian to conclude the properties of their evolution. Also we have stored more snapshot from the same versions of these distributions, so we can analyse the differences of the versions and the differences of the same version in different times as well.

A part of the package dependency network of the Ubuntu 11.10 stored on June 15, 2012 can be seen in Fig. 1, the package of the Vim text editor with its neighbours. At left there are the packages, that need the Vim package installed (the packages that depends on Vim), and at right the packages that need to install before the installation of Vim (the packages that depends Vim on). There are two types of packages. Most packages, like Vim, are real packages. They can be downloaded and installed. In Fig. 1 these are plotted with red circles. There are some virtual packages, like `editor` in the figure. These packages can be among the dependencies of the real packages, and at least one package should state that it provides this package. In the figure the packages `vim`, `vim-gnome` provide that packages, and some others that can not be seen in the picture like `emacs`, `nano` and `mcedit`, so if one package lists editor as its dependency, than one of these packages is enough to be installed, but we do not need to install the favourite editor of the package maintainer. In the picture, there are blue arrows from the virtual package to the packages that provide it. The dependencies are plotted with gold arrows. The red arrows in the pictures means, that the package at the start of the arrow recommends the package at the head of the arrow. There are other connection types and source packages [6], that are not discussed here and are not stored yet in our network archives.

The package dependency network of the Ubuntu 11.10 dated June 15, 2012 has 36685 (88.7 %) real packages and 4686 (11.3 %) virtual packages. Among them there are 170185 connections: 150249 (88.3 %) dependencies, 13077 (7.7 %) recommendations and 6859 (4.0 %) provisions. There is no selfloop in the network, but there are multiple edges: at least 566 edges (0.3 %) need to remove from the network not to have multiple edges. A typical situation with multiple edges is that one of the edges is dependency and the other is recommendation: in the simplest variaton one package can depends on Python 2.5 but recommends Python 2.6. In the package network of the Ubuntu 12.04 dated June 1, 2012 there are 414 selfloops. All of them are provisions. Some of the nodes with selfloops are: dpkg, xz-utils, coreutils, tar, m4.

Our earlier article investigates the software package network of the Jaunty (9.04) version of Ubuntu distribution dated November 3, 2009. That network does not include the virtual packages, provisions and recommendations, just real packages with dependencies.

In that network the number of nodes and were $N = 27554$ and $M = 126540$ respectively. The network was not connected, but 93.14% of the nodes belonged to one, the largest component. The diameter of the largest component was 13.

The cumulative degree distribution of the network can be seen in Fig. 2. We can see without any calculation, that the scale-free model better fits the distribution than the random graph model.

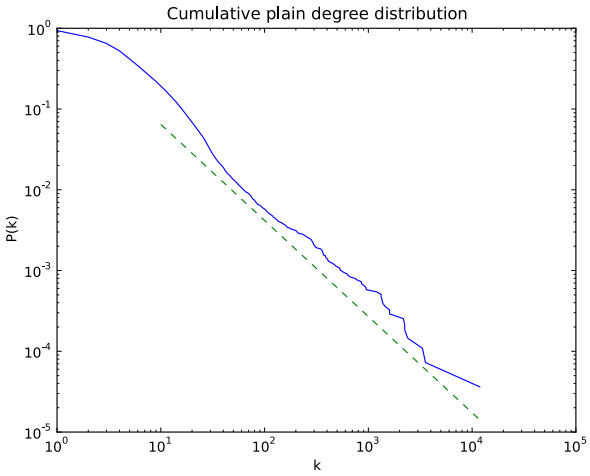

Figure 2. The cumulative degree distribution of the package dependency network (continuous) and the power-law function with the exponent $-1,19$ given by the maximum likelihood method (dashed). $\left(\gamma = 2,19 \pm 0,14\,^{+0,1}_{-0,12}\right)$

If we assume the degree distribution of the network is power-law, the absolute value of its exponent, $\gamma > 0$ can be estimated with the maximum likelihood method. The slightly modified version of the estimate for continuous power-law distributions can also be used as a good approximation for discrete power-law distributions [7], [8]. This modified equation is:

$$\gamma = 1 + N \left[\sum_{k_i > k_{\min}} \ln \frac{k_i}{k_{\min} - 0.5} \right]^{-1} \qquad k_{\min} \in \mathbb{Z}\,, \qquad (1)$$

where $k_{\min}$ means the minimum value of degrees above which we assume the power-law behaviour. The estimate for its standard deviation is

$$\sigma = \frac{\gamma - 1}{\sqrt{N}}\,. \qquad (2)$$

We have found the degree distribution a power-law function for degrees larger than 240 with the exponent

$$\gamma = 2.19 \pm 0.14\,^{+0.1}_{-0.12}\,.$$

We can find the nodes (packages) with the largest in- and out degrees, and we can compare them with the average of the in- and out-degrees. Table I compares three states of the network coming from three distinct dates and versions. There were notable changes in the order of the packages, and there were nodes, with decreased degrees as well.

The average of the in-degree can be calculated as

$$\langle k_{\text{in}} \rangle = M/N = 4.592.$$

However the largest in-degree (11868) was more than 2500 times larger.

[1]The name of the package was libqt4-core in the version 7.10

Table I
THE PACKAGES WITH THE LARGEST IN-DEGREE ($k_{\rm in}> 1000$) FOR THREE SNAPSHOT OF THE PACKAGE NETWORK. $k_{\rm in}$: ON JULY 10, 2012 (VER. 11.10), $k_{\rm in}$': ON NOVEMBER 3, 2009 (VER. 9.04), $k_{\rm in}$": ON AUGUST 29, 2008 (VER. 7.10).

$k_{\rm in}$	$k_{\rm in}$'	$k_{\rm in}$"	package	description
14565	11866	10748	libc6	C shared libraries
4056	3319	2803	libstdc++6	Standard C++ library
3779	3548	2970	libgcc1	Libraries of C compiler
3397	2229	1842	perl	Perl language
2827	1595	1122	python	Python language
2684	2243	1818	libglib2.0-0	The GLib library
1571	2170	3172	libx11-6	X11 client-side library
1457	947	945	dpkg	deb package management
1435	1571	1275	libgtk2.0-0	The GTK+ graphical user interface library
1217	553	108	libqtcore4 [1]	Qt 4 core module
1189	2399	1963	zlib1g	Compression library
1018	666	447	python-support	Automated rebuilding support for Python modules

Table II
PACKAGES WITH THE LARGEST OUT-DEGREE ($k_{\rm out}\geq 90$). $k_{\rm out}$: ON JULY 10, 2012, $k_{\rm out}$': ON NOVEMBER 3, 2009. WHERE $k_{\rm out}$' IS MARKED WITH – (DASH) THERE WAS NOT A PACKAGE WITH THE SAME NAME.

$k_{\rm out}$	$k_{\rm out}$'	package	description
210	–	ubuntu-sugar-remix	Programs for the education
200	113	ubuntu-desktop	Ubuntu with GNOME desktop environment
199	74	xubuntu-desktop	Ubuntu with XFCE desktop environment
176	111	ubuntustudio-desktop	Multimedia creation flavor of Ubuntu
176	66	kubuntu-desktop	Ubuntu with KDE desktop environment
144	133	ichthux-desktop	Desktop for Christians
130	–	libmono-cil-dev	Development files for Mono
119	–	kubuntu-full	The Kubuntu with additional packages.
106	103	texlive-full	Meta package pulling in all components of TeX Live
102	2	med-bio	Packages for biology
97	83	libjifty-perl	Programkönyvtár a Jifty web-keretrendszerhez
90	–	lubuntu-desktop	Desktop with LXCE desktop environment

The clustering coefficient is defined for undirected networks, so we have to transform our directed network into undirected. The clustering coefficient of the network is 0.308. We can compare it with the values coming from network models. If the network were a random graph, the p probability of edges, as well as the C clustering coefficient would be

$$p = \frac{M}{N(N-1)/2} = C = 0.000333. \qquad (3)$$

In the Barabási–Albert model the mean degree is two times the number of new edges in one step: $2m$. As the mean degree is

$$\langle k\rangle = 2M/N = 9.1849, \text{ so the best parameter is } m = 5$$

for the corresponding Barabási–Albert model. With the parameters $N = 27554$, $m = 5$ we obtained a network with 137745 edges and clustering coefficient 0.0032. The parameters of the two models give us a clustering coefficient 2–3 orders of magnitude smaller, than that of the package dependency network. The clustering coefficient of the hierarchical model is $C_h = 0.743$ [9], which is in the same order of magnitude as the coefficient of the dependency network.

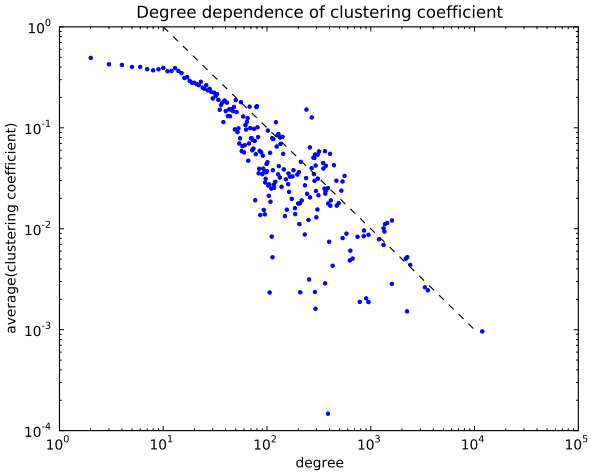

Figure 3. The dependence of the average clustering coefficient on the degree (dots) and the power law function with the exponent -1 (dashed line)

Plotting the clustering coefficient of the nodes with given degree as a function of degree (Fig. 3) one can conclude, that the function does not contradict the degree dependency of other networks [9]. The dashed line is the line of the power law function with the exponent -1. So the package dependency network can be called hierarchical network.

III. CHANGES IN THE IN-DEGREE

For the Ubuntu Linux we have compared the in-degrees of two versions. We have chosen the packages that were present in both version and we plot the changes of the in-degrees as a function of the in-degree in the earlier version in Fig. 4. We have counted all of the three types of edges (dependency, provision and recommendation).

The scatter plot confirms the existence of an approximate proportionality between Δk_{in} and k_{in} in a good agreement with [10]. This proportionality is the base of more network evolution model like the Barabási–Albert model and the Price model.

IV. THE ARCH LINUX

We have made a snapshot from the sofware package network of the Arch Linux. This snapshot was made on April 25, 2012. The network has $N = 4047$ nodes and $M = 11524$ dependencies, so its average degree was $\langle k\rangle = 5.695$. The maximal degree was $k_{max} = 406$. The biggest weakly connected component includes 3643 nodes (90 % of the nodes), the others are isolated nodes. The biggest strongly connected component has only 2 nodes (udev and util-linux depended to each another), the others have one nodes per component. So this network has no more than one cycle.

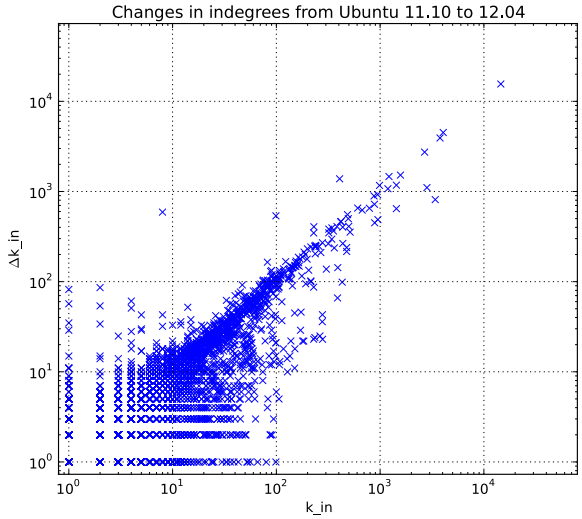

Figure 4. The changes of the in-degree of the packages of Ubuntu as a function of the original in-degree in double logarithmic scale. The original version is 11.10 (June 5, 2012), and the other version is 12.04 (June 13, 2012)

V. Conclusion

We studied the software package network of the Ubuntu, Debian and Arch Linux distributions. We investigated three types of connections between packages: dependencies, recommendations and provisions. The software package dependency network is a scale-free hierarchical network, sharing some properties with other real networks. In this network the nodes (the packages) with largest indegree tends to get more connections, that the nodes with smallest one.

References

[1] [Online]. Available: http://www.debian.org/

[2] "python-apt documentation." [Online]. Available: http://apt.alioth.debian.org/python-apt-doc/

[3] "cxnet homepage." [Online]. Available: http://django.arek.uni-obuda.hu/cxnet/

[4] A. Horváth and Z. Trócsányi, "Complex networks in the curriculum of computer engineers," *IEEE Proceedings of the 8th International Symposium on Applied Machine Intelligence and Informatics*, 2010.

[5] A. Horváth, "Studying complex networks with cxnet," *Acta Physica Debrecina*, vol. XLIV, 2010.

[6] "Basics of the debian package management system." [Online]. Available: http://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html

[7] M. E. J. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, p. 323, 2005. [Online]. Available: http://arxiv.org/abs/cond-mat/0412004

[8] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," 2007. [Online]. Available: http://arxiv.org/abs/0706.1062

[9] E. Ravasz and A. Barabasi, "Hierarchical organization in complex networks," *PHYSICAL REVIEW E*, vol. 67, no. 026112, FEB 2003.

[10] T. Maillart, D. Sornette, S. Spaeth, and G. von Krogh, "Empirical tests of Zipf's law mechanism in open source Linux distribution," *Phys. Rev. Lett.*, vol. 101, p. 218701, 2008. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.101.218701