Physica A 447 (2016) 520–526

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Network of networks in Linux operating system

Haoqin Wang[a,b], Zhen Chen[c], Guanping Xiao[b], Zheng Zheng[a,b,*]

[a] State Key Laboratory of Software Development Environment, Beijing, 100191, PR China
[b] School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, PR China
[c] School of Electronic and Information Engineering, Beihang University, Beijing 100191, PR China

HIGHLIGHTS

- A network of networks about Linux operating system is constructed.
- Degree distribution similarity among the whole and component networks is observed.
- Manifestations in topology and function for the components have been observed.
- System failures make some nodes unreachable and visit new nodes at the same time.

ARTICLE INFO

Article history:
Received 13 September 2015
Received in revised form 26 October 2015
Available online 29 December 2015

Keywords:
Complex network
Linux operating system
Coupling relationship
System failure

ABSTRACT

Operating system represents one of the most complex man-made systems. In this paper, we analyze Linux Operating System (LOS) as a complex network via modeling functions as nodes and function calls as edges. It is found that for the LOS network and modularized components within it, the out-degree follows an exponential distribution and the in-degree follows a power-law distribution. For better understanding the underlying design principles of LOS, we explore the coupling correlations of components in LOS from aspects of topology and function. The result shows that the component for device drivers has a strong manifestation in topology while a weak manifestation in function. However, the component for process management shows the contrary phenomenon. Moreover, in an effort to investigate the impact of system failures on networks, we make a comparison between the networks traced from normal and failure status of LOS. This leads to a conclusion that the failure will change function calls which should be executed in normal status and introduce new function calls in the meanwhile.

1. Introduction

A wide range of real-world systems, such as the World Wide Web [1,2], scientific collaborations [3] and transportation infrastructures [4,5], can be described as complex networks where the entities are denoted by nodes and the relationships between entities are denoted by edges. In 1959, Erdös and Rényi introduced ER random network model [6], which has dominated the research on the complex network for decades. After that, the proposal of small-world models by Watts et al. [7] and scale-free properties by Barabási et al. [8] has attracted more and more researches on complex network in a wide field until now. In recent years, the researches on complex networks are flourishing, such as network modeling [9–11], epidemic spreading [12,13], cascading failures [14], traffic dynamics [15], evolutionary games [16–20], optimization process [21,22] and social dynamics [23–25]. One interesting direction is to study software systems [26–28] from the view of complex networks.

* Corresponding author.
E-mail address: zhengz@buaa.edu.cn (Z. Zheng).

http://dx.doi.org/10.1016/j.physa.2015.12.084
0378-4371/© 2015 Elsevier B.V. All rights reserved.

Table 1
Basic properties of the six components. n is the number of nodes; m is the number of edges; $\langle k_{out} \rangle$ is the average out-degree; $\langle k_{in} \rangle$ is the average in-degree; $\langle f_{out} \rangle$ is the average external out-links and equals to the ratio of the number of external out-links to n; $\langle f_{in} \rangle$ is the average external in-links and equals to the ratio of the number of external in-links to n; C is the average clustering coefficient; E is the efficiency [41].

	arch	*drivers*	*fs*	*kernel*	*mm*	*net*
n	41 535	221 837	23 955	7656	3173	23 430
m	46 220	367 986	34 197	6494	3779	41 389
$\langle k_{out} \rangle$	1.112796	1.658813	1.427552	0.848224	1.190986	1.766496
$\langle k_{in} \rangle$	1.112796	1.658813	1.427552	0.848224	1.190986	1.766496
$\langle f_{out} \rangle$	2.276851	2.516708	4.250637	2.898511	3.453514	2.908451
$\langle f_{in} \rangle$	5.524088	0.532256	6.004258	13.74125	31.39994	6.820401
C	0.031023	0.043741	0.042347	0.029811	0.029721	0.035559
E	0.000495	0.000081	0.000733	0.001519	0.004204	0.000283

Software systems represent one of the most complex man-made systems and can be expressed as networks [29]. A wealth of study on software systems from the prospective of complex networks has been conducted over the past decade. Valverde et al. [30] presented the first evidence for the emergence of scaling and the presence of small world behavior in software systems. Myers [31] examined software systems as complex networks and provided a model of software evolution based on refactoring processes. Cai et al. [32] proposed a software mirror graph to record the dynamic information of software behaviors.

Operating system (OS) is one of the most important software systems and provides a basic executing environment for other software. Among various operating systems, Linux operating system (LOS), which is first released by Linus Torvalds in 1991, is well deployed in nearly all fields of our society nowadays. Understanding the inner structure of LOS is helpful for its maintenance and development. Additionally, the source open characteristic of LOS makes it convenient to analyze a software system as a complex network. In 2008, Zheng et al. [33] put forward two network growth models to better describe the properties of LOS. Recently, Gao et al. [34] analyzed the core components of LOS as a network and observed a scale-free phenomenon in it.

Furthermore, since LOS consists of several independent components, the interactions of which are essential for the execution of LOS. Thus, it is necessary and possible to explore the coupling relationships of components. The study of network of networks is a hot topic these days [35], and researchers had found that many realistic systems are interdependent on other systems. In 2010, Buldyrev et al. [36] developed a framework for analyzing the coupling relations of two interdependent networks. Whereafter, coupling relations have been extensively studied [37–39].

In this paper, we study the network of networks in LOS to explore the coupling relationships among components. The rest of this paper is organized as follows. Section 2 proposes the network modeling of LOS and the analysis of topological properties for six modularized components in LOS. Section 3 presents the coupling relationships of components both in topology and function. In Section 4, the impacts of system failures on LOS network are discussed. Finally, the conclusion of the work is given in Section 5.

2. Network modeling of LOS

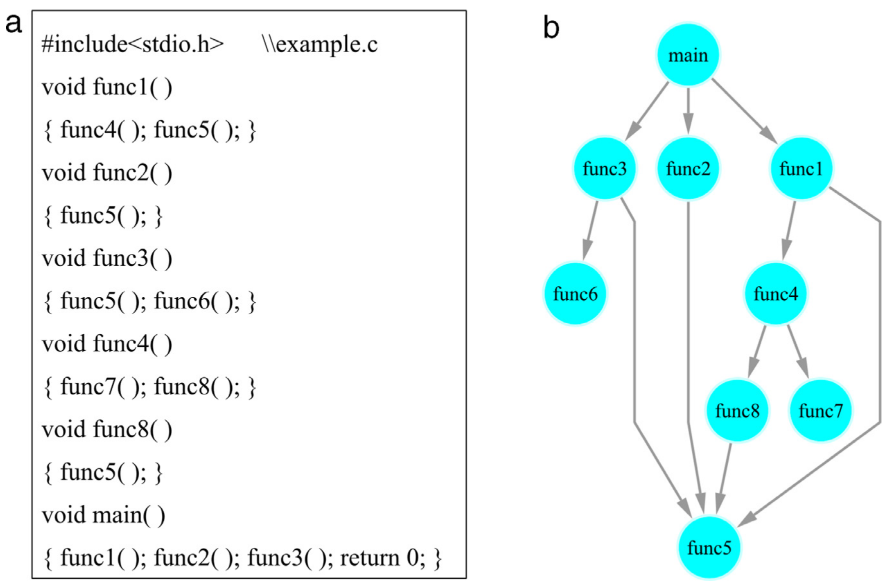
LOS is mainly composed of six interacting components: *arch*, *drivers*, *fs*, *kernel*, *mm* and *net*. *arch* determines the feasible hardware that LOS can be installed in; *drivers* contains the device drivers; *fs* is the component of the disk and file system; *kernel* manages the processes in LOS; *mm* is the component of memory management and *net* serves networking [40]. We model LOS (Linux-3.16.1[1]) as a directed network, in which nodes represent functions and edges represent function calls. Fig. 1 illustrates a simple example for the network modeling.

It should be noted that the network of each component only contains the calls between functions of the same component. Additionally, the whole LOS network is constructed by component networks through the function calls between them.

Now we study the topological properties of the six components. It can be observed from Table 1 that the numbers of nodes for the components are quite different from each other. Among the six components, *drivers* is the largest one which is about 70 times larger than the smallest, i.e. *mm*. Each component has external links to communicate with other components. It shows that the value of $\langle f_{out} \rangle$ in *fs* is the largest, indicating that the realization of *fs* needs more interactions with other components. Moreover, the values of $\langle f_{in} \rangle$ in *kernel* and *mm* are larger, indicating that the functions in *kernel* and *mm* are called by other components more significantly. Another point worth mentioning is the efficiency. From the table, we can observe that the efficiency of *drivers* is extremely low. This is ascribed to two reasons. First, it is hard to find a loopback in the LOS network because there is no mutual function call between two functions in LOS. Besides, the number of nodes in *drivers* is the largest, which can account for the lowest efficiency in *drivers* dominantly. In the following, we will discuss the degree distributions of networks.

Fig. 2 exhibits the degree distribution of the whole system and the six components, which denotes the probability that the in-degree or out-degree of a randomly selected node is k. The figure indicates that the in-degree (or out-degree)

[1] We choose Linux-3.16.1 as our analyzing object, and it was the newest version when we started this research.



```
#include<stdio.h>      \\example.c
void func1( )
{ func4( ); func5( ); }
void func2( )
{ func5( ); }
void func3( )
{ func5( ); func6( ); }
void func4( )
{ func7( ); func8( ); }
void func8( )
{ func5( ); }
void main( )
{ func1( ); func2( ); func3( ); return 0; }
```

Fig. 1. An illustration of network modeling. (a) Is a piece of C language program; (b) Is a directed network based on the function calls in (a). Each function maps to a node and each function call is represented by an edge. Note, the in-degree of a function represents the number of functions calling it, e.g. func5's in-degree is 4. The out-degree of a function represents the number of functions it calls, e.g. func4's out-degree is 2.

distributions of the LOS network and the six components are close to each other. In addition, we find that the out-degree distribution follows an exponential distribution and the in-degree distribution follows a power-law distribution, which is in well accordance with the results of Gao et al. [34]. Furthermore, it can be found that the maximal out-degree is much smaller than the maximal in-degree for all components. The reason is that, in general, professional programmers prefer to write a function with only a few calls to make it reliable and readable when programming, which limits the value of the maximal out-degree. Meanwhile, the functions providing basic services are called by huge number of high-level functions and this is why they have extremely big in-degree, especially for the functions in large components.

3. Network of networks in LOS

The execution of LOS requires the interactions among components, which can be described as coupling relationships in networks. There are two types of manifestations in coupling relationships, topological and functional. The topological coupling relationships represent *all* the connections between components and reflect the underlying design principles of LOS to some extent. Note that, during the execution of LOS, not all of the connections are covered. The functional coupling relationships only cover the connections which are utilized by the *executed* components, and they reflect the real-time execution status of LOS. It is a subset of the topological coupling relationships. In the remaining of this section, we design an experiment to study both the topological and functional coupling relationships.

The topological coupling relationships can be obtained from the source code of LOS by static analyzing and the functional coupling relationships can be traced by *ftrace*,[2] a tool provided by LOS. *ftrace* can trace the function calls in the execution of LOS. We run several open and authoritative benchmarks[3] of Linux to emulate users operations and cover functions as many as possible.

Fig. 3(a) illustrates the topological coupling relationships of the six components. It is observed that all components have the largest number of coupling edges connecting to *drivers*, which means *drivers* is the key component from the view of topology. However, when the LOS is running, *kernel* becomes the most crucial component from the view of function, as shown in Fig. 3(b). The significant contrast between topological and functional coupling relationships can be explained as follows. The compatibility of devices is one of the key principles when designing an operating system, LOS as an example. Component *drivers* contains extremely huge number of device drivers with different types, which are requested by the interactions between other components and devices. Hence, the number of coupling edges, of which each component connects to *drivers*, is the largest from the prospective of topology. However, during the execution of LOS, the running devices are specific and limited. This leads to the weak manifestation of *drivers* in function. In addition, the execution of LOS requires the cooperation of components which are managed by *kernel*. Therefore, *kernel* has a strong manifestation and becomes the most crucial component when LOS is running.

2 http://elinux.org/Ftrace.
3 The benchmarks we use are downloaded from http://www.phoronix-test-suite.com and https://code.google.com/p/byte-unixbench.

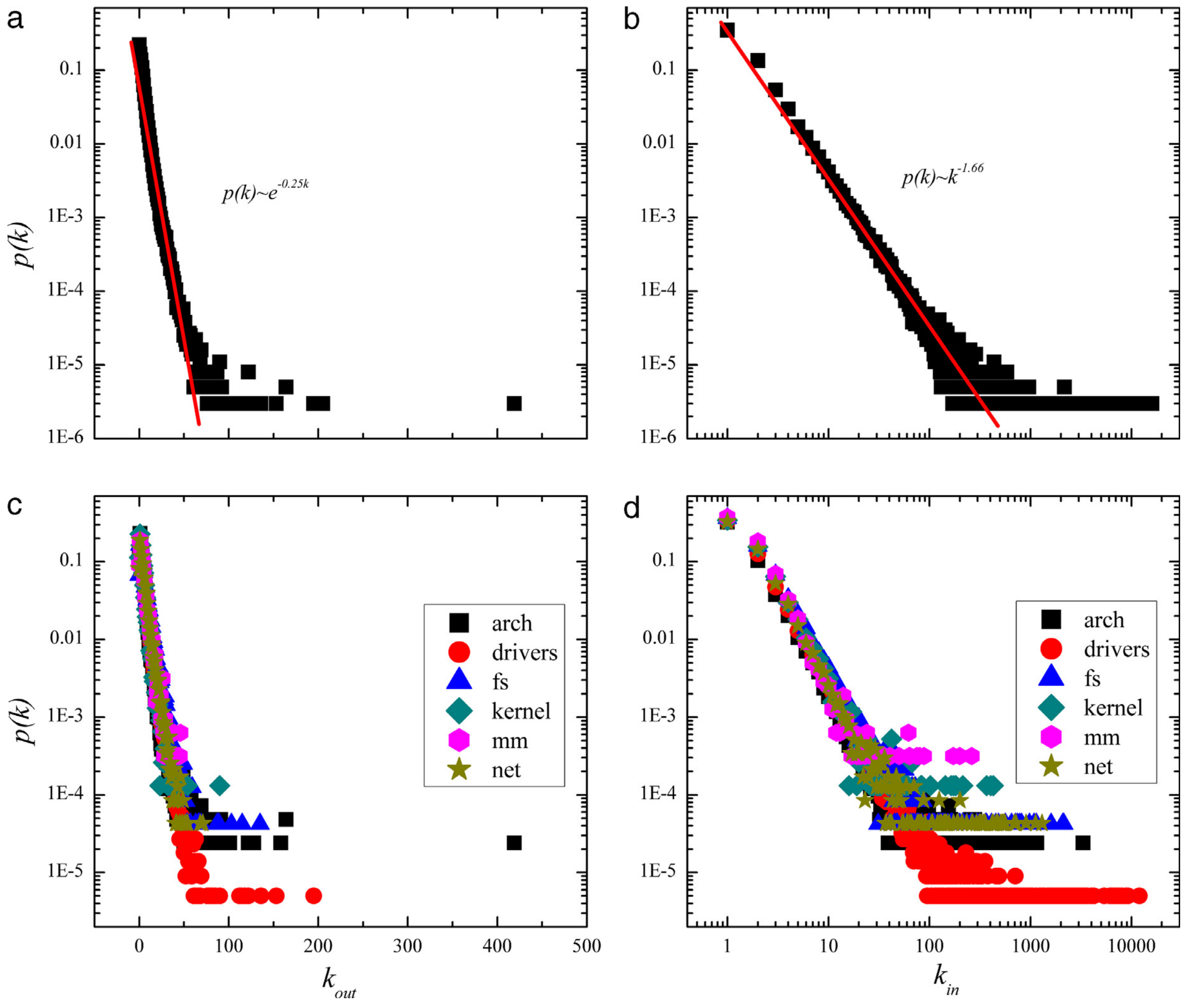


Fig. 2. The degree distributions of networks. k_{out} is the out-degree and k_{in} is the in-degree. (a) Out-degree distribution of the LOS network; (b) in-degree distribution of the LOS network; (c) out-degree distribution of the six components; (d) in-degree distribution of the six components. The red lines in (a) and (b) are fitting lines and the slopes of them indicating that the out-degree distribution follows $p(k) \sim e^{-0.25k}$ and in-degree distribution follows $p(k) \sim k^{-1.66}$, respectively. (Note that, the degree distributions of the whole network and the components are similar.) (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

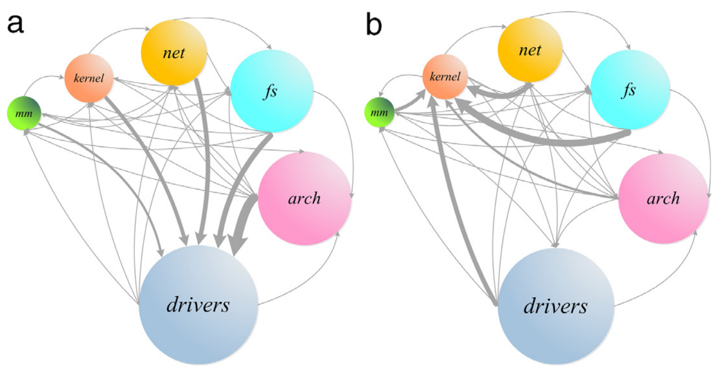


Fig. 3. An illustration of network of networks in LOS. (a) The topological coupling relationships of the six components; (b) the functional coupling relationships of the six components. Each node in (a) or (b) represents a component. Each edge represents that there exist function calls between two components and edge thickness represents interaction strength.

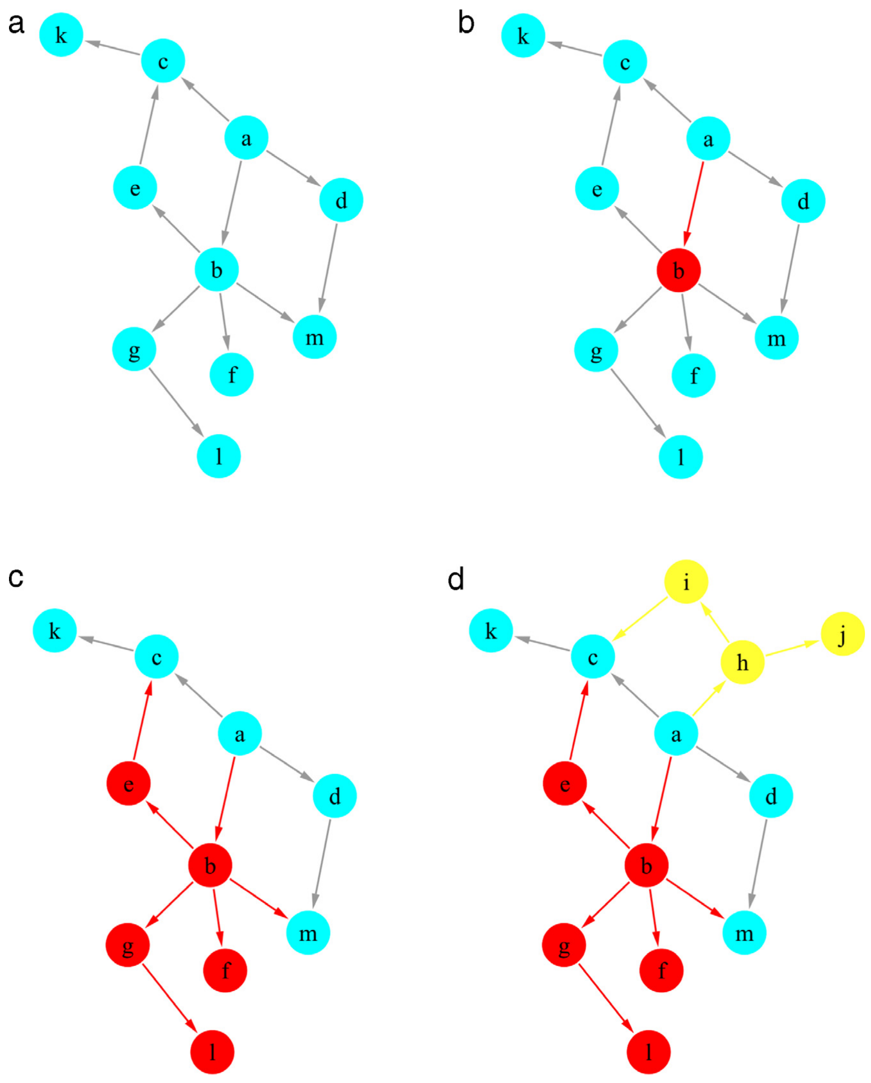


Fig. 4. An example to show the failure process. (a) The normal executing status. (b) A function call fails and this leads to the invalid of node b. (c) The nodes which are only called by invalid nodes become invalid, e.g. the node f; Otherwise, the nodes still work, e.g. the node c. (d) New nodes are visited to respond the failure, e.g. the nodes i, j and h.

4. The impact of system failures on network

LOS is a giant and complicated software system. During its execution time, system failures are inevitable. In this section, to explore the impact of system failures on network, we design an experiment to make a comparison between normal and failure status of LOS.

The procedures of the experiment can be briefly described as follows. We injected bugs in LOS first, and then triggered them while the LOS was executing. Due to the huge number of fixed bugs during the long term maintenance and evolution of LOS, we utilize the following constraints for the injection of bugs to limit them to an appropriate number.

(1) the bugs are fixed in the period of released time from version 3.15 to version 3.16, due to the specific research version in this paper;
(2) the bugs can be reproduced in our current computer environment.[4]

After filtering by the constraints, there are totally 27 bugs injected and we denote the new generated LOS as LOS_{bug}. We used all benchmarks mentioned in Section 3 to trigger the bugs in LOS_{bug} randomly to avoid the influence of human factors, then traced function calls and built a network named NF. Those benchmarks were again executed in the LOS (without bugs)

[4] A Lenovo laptop with: CPU Intel Core i5-3230; 4 GB of memory; 500 GB of hard disk; Ubuntu-14.04 with the version of 3.16.1 Linux kernel.

and a network named NS was constructed by the similar way. There are 6507 nodes in NS and 6090 nodes in NF. To explore the impact of system failures on network, we compared the nodes and edges in NF and NS carefully. The discrepancies between NF and NS are caused by the system failures.

With the comparison of the nodes and edges in NS and NF, we find that the failures not only reduce the nodes in NS, but also insert new nodes into NF. This phenomenon could be the result of the system interruption caused by abnormal executions, wrong judgments of predicates and so on. Fig. 4 illustrates the procedure of a system interruption caused by an abnormal execution in LOS. Fig. 4(a) depicts the normal execution status. Once a failure occurs and is traced by LOS, as shown in Fig. 4(b), the LOS will call system interruption process and interrupt the current process. Due to the interruption, a large number of functions will be unreachable, as indicated in Fig. 4(c). Meanwhile, additional functions (such as a warning information to users) corresponding to the interruption are executed as shown in Fig. 4(d). The interruption stops the propagating of the failure and can be regarded as a manifestation of the system robustness.

5. Conclusion

In this paper, we investigate LOS from the view of network of networks. We observe that the six interacting components have similar degree distributions with the whole LOS network, in which the out-degree follows an exponential distribution and the in-degree follows a power-law distribution. The topological and functional coupling relationships between components are analyzed, and it has been found that component *drivers* has a strong manifestation in topology while a weak manifestation in function, but the case of component *kernel* is on the contrary. Moreover, an experiment was designed to investigate the impact of system failures on network. We find that the failure will reduce a lot of functions and introduce another part of functions in the meanwhile. Our work presents a way to better understand the network of networks in LOS and may shed light on the bug detection in network of software systems.

Acknowledgment

This work is supported by the State Key Laboratory of Software Development Environment. And the authors would like to thank Dr. Wen-Bo Du from Beihang University for kindly providing valuable technical advices to this manuscript.

References

[1] R. Pastor-Satorras, A. Vázquez, A. Vespignani, Dynamical and correlation properties of the Internet, Phys. Rev. Lett. 87 (2001) 258701.
[2] R. Albert, H. Jeong, A.L. Barabási, Internet: Diameter of the world-wide web, Nature 401 (1999) 130.
[3] M.E.J. Newman, Scientific collaboration networks. I. Network construction and fundamental results, Phys. Rev. E 64 (2001) 016131.
[4] G. Bagler, Analysis of the airport network of india as a complex weighted network, Physica A 387 (2008) 2972.
[5] J. Zhang, X.B. Cao, W.B. Du, K.Q. Cai, Evolution of Chinese airport network, Physica A 389 (2010) 3922.
[6] P. Erdős, A. Rényi, On random graphs I, Publ. Math. Debrecen 6 (1959) 290.
[7] D.J. Watts, S.H. Strogatz, Collective dynamics of small-world networks, Nature 393 (1998) 440.
[8] A.L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509.
[9] A. Barrat, M. Barthélemy, A. Vespignani, Weighted evolving networks: coupling topology and weight dynamics, Phys. Rev. Lett. 92 (2004) 228701.
[10] T. Zhou, G. Yan, B.H. Wang, Maximal planar networks with large clustering coefficient and power-law degree distribution, Phys. Rev. E 71 (2005) 046141.
[11] W.X. Wang, B.H. Wang, B. Hu, G. Yan, Q. Ou, General dynamics of topology and traffic on weighted technological networks, Phys. Rev. Lett. 94 (2005) 188702.
[12] R. Pastor-Satorras, A. Vespignani, Epidemic spreading in scale-free networks, Phys. Rev. Lett. 86 (2001) 3200.
[13] H.X. Yang, W.X. Wang, Y.C. Lai, Y.B. Xie, B.H. Wang, Control of epidemic spreading on complex networks by local traffic dynamics, Phys. Rev. E 84 (2011) 045101.
[14] W. Wang, Y.C. Lai, Abnormal cascading on complex networks, Phys. Rev. E 80 (2009) 036109.
[15] W.B. Du, Z.X. Wu, K.Q. Cai, Effective usage of shortest paths promotes transportation efficiency on scale-free networks, Physica A 392 (2013) 3505.
[16] W.B. Du, X.B. Cao, M.B. Hu, W.X. Wang, Asymmetric cost in snowdrift game on scale-free networks, Europhys. Lett. 87 (2009) 60004.
[17] Z. Wang, A. Szolnoki, M. Perc, Optimal interdependence between networks for the evolution of cooperation, Sci. Rep. 3 (2013) 2470.
[18] Z. Wang, A. Szolnoki, M. Perc, Different perceptions of social dilemmas: Evolutionary multigames in structured populations, Phys. Rev. E 90 (2014) 032813.
[19] Z. Wang, L. Wang, A. Szolnoki, M. Perc, Evolutionary games on multilayer networks: a colloquium, Eur. Phys. J. B 88 (2015) 1.
[20] A. Szolnoki, M. Perc, Reentrant phase transitions and defensive alliances in social dilemmas with informed strategies, Europhys. Lett. 110 (2015) 38003.
[21] C. Liu, W.B. Du, W.X. Wang, Particle swarm optimization with scale-free interactions, PLoS One 9 (2014) e97822.
[22] Y. Gao, W.B. Du, G. Yan, Selectively-informed particle swarm optimization, Sci. Rep. 5 (2015) 9295.
[23] C. Castellano, S. Fortunato, V. Loreto, Statistical physics of social dynamics, Rev. Modern Phys. 81 (2009) 591.
[24] Z. Wang, Y. Liu, L. Wang, Y. Zhang, Freezing period strongly impacts the emergence of a global consensus in the voter model, Sci. Rep. 4 (2014) 3597.
[25] W.B. Du, Y. Gao, C. Liu, Z. Zheng, Z. Wang, Adequate is better: particle swarm optimization with limited-information, Appl. Math. Comput. 268 (2015) 832.
[26] S. Jenkins, S.R. Kirk, Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution, Inform. Sci. 177 (2007) 2587.
[27] G. Concas, M. Marchesi, S. Pinna, N. Serra, Power-laws in a large object-oriented software system, IEEE Trans. Softw. Eng. 33 (2007) 687.
[28] P. Louridas, D. Spinellis, V. Vlachos, Power laws in software, ACM Trans. Softw. Eng. Methodol. 18 (2008) 2.
[29] A. Potanin, J. Noble, M. Frean, R. Biddle, Scale-free geometry in OO programs, Commun. ACM 48 (2005) 99.
[30] S. Valverde, R.F. Cancho, R.V. Sole, Scale-free networks from optimal design, Europhys. Lett. 60 (2002) 512.
[31] C.R. Myers, Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, Phys. Rev. E 68 (2003) 046116.
[32] K.Y. Cai, B.B. Yin, Software execution processes as an evolving complex network, Inform. Sci. 179 (2009) 1903.
[33] X.L. Zheng, D. Zeng, H.Q. Li, F.Y. Wang, Analyzing open-source software systems as complex networks, Physica A 387 (2008) 6190.

[34] Y.C. Gao, Z. Zheng, F.Y. Qin, Analysis of linux kernel as a complex network, Chaos Solitons Fractals 69 (2014) 246.
[35] W.B. Du, X.L. Zhou, Y.B. Zhu, Z. Zheng, A dynamic allocation mechanism of delivering capacity in coupled networks, Chaos Solitons Fractals 80 (2015) 56.
[36] S.V. Buldyrev, R. Parshani, G. Paul, H.E. Stanley, S. Havlin, Catastrophic cascade of failures in interdependent networks, Nature 464 (2010) 1025.
[37] J. Shao, S.V. Buldyrev, S. Havlin, H.E. Stanley, Cascade of failures in coupled network systems with multiple support-dependence relations, Phys. Rev. E 83 (2011) 036116.
[38] F. Tan, Y.X. Xia, W.P. Zhang, X.Y. Jin, Cascading failures of loads in interconnected networks under intentional attack, Europhys. Lett. 102 (2013) 28009.
[39] W.B. Du, X.L. Zhou, Z. Chen, K.Q. Cai, X.B. Cao, Traffic dynamics on coupled spatial networks, Chaos Solitons Fractals 68 (2014) 72.
[40] R. Love, Linux Kernel Development, Pearson Education, 2010.
[41] V. Latora, M. Marchiori, Efficient behavior of small-world networks, Phys. Rev. Lett. 87 (2001) 198701.