Analysis of Linux kernel as a complex network

Yichao Gao, Zheng Zheng *, Fangyun Qin

School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China

A R T I C L E   I N F O

Article history:
Received 17 June 2014
Accepted 14 October 2014
Available online 9 November 2014

A B S T R A C T

Operating system (OS) acts as an intermediary between software and hardware in computer-based systems. In this paper, we analyze the core of the typical Linux OS, Linux kernel, as a complex network to investigate its underlying design principles. It is found that the Linux Kernel Network (LKN) is a directed network and its out-degree follows an exponential distribution while the in-degree follows a power-law distribution. The correlation between topology and functions is also explored, by which we find that LKN is a highly modularized network with 12 key communities. Moreover, we investigate the robustness of LKN under random failures and intentional attacks. The result shows that the failure of the large in-degree nodes providing basic services will do more damage on the whole system. Our work may shed some light on the design of complex software systems.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Many natural and technological systems, such as the Internet [1–3], social networks [4–6], biological networks [7–9], and airline networks [10–12], can be represented as complex networks. These networks consist of a set of nodes representing entities of the systems connected by edges indicating relationships between entities. In 1950s, Erdős and Rényi introduced the famous ER random network model [13], which has guided the research on complex networks for decades. Since the small-word network model [14] and scale-free network model [15] were proposed at the end of last century, it is found that many real-world networks are actually complex networks associated with small-world properties and power-law degree distributions. In recent years, the study of complex networks has attracted various scientific communities, such as network modeling [16–18], synchronization [19,20], cascading failures [21,22], evolutionary games [23–25], optimization [26], network traffic [27,28] and epidemic spreading [29,30]. One interesting research direction is to explore software systems in the framework of complex network theory [31–33].

As the centerpiece of information world, software systems play more and more important roles in our daily life. How to build high quality software systems is one of our major concerns. In the last decades, the software engineering, whose objective is to provide methodologies and tools to efficiently design and build software systems [34], has been widely introduced. However, it remains a challenge to model software system due to its high complexity. Software systems are typical man-made systems which can be represented as networks. Valverde et al. [35] presented the first evidence for the emergence of scale-free and small-world structure in software networks. Cai and Yin [36] proposed the software mirror graph to record the runtime information of a software system under given environments. Gorshenev and Pis’mak [37] investigated the evolution of software systems and proposed a model representing the natural selection principle behind software evolution. The studies of software using graph methods are helpful to predict defects of software systems and to develop large software systems with well-designed structures.

Among various software systems, operating system (OS) is the foundation of other kind of software systems.
It provides interfaces to software and hardware, allowing users to communicate with computers. There are a lot of popular OSs such as Microsoft Windows, Linux, OS X, and UNIX, in which Linux is a well-known open source OS.¹ Thus, ever since it was first released by Linus Torvalds in 1991, Linux has been arousing millions of volunteers’ interests to add, change and improve the system. Nowadays, it is a matured OS widely utilized, ranging from cell phones to supercomputers. In this paper, we will study the Linux system from the network point of view. For the high complexity of Linux, we focus on the kernel, which is the innermost part of Linux.

The rest of this paper is organized as follows. Section 2 presents LKN’s topological properties to investigate the underlying design principles, followed by the analysis of LKN’s community structures and its relationship with kernel’s functional modules in Section 3. In Section 4, the robustness of LKN is discussed. Section 5 concludes this study.

2. Topological properties of LKN

Linux kernel consists of thousands of functions which collaborate with each other through function calls. For example, Fig. 1 illustrates the function call relationship among 5 functions: foo1 calls foo2 and foo3, foo2 calls foo4, and foo3 calls foo5. With functions as nodes and function calls as edges, the relationship among the 5 functions can be represented as a directed network. Note that, Linux is updated almost every 2 months and there have been hundreds of releases until now. In this research, we select the latest version (v3.12.6) as our object. We build up the LKN as a directed complex network and focus on its largest weakly connected component, which consist of 9334 nodes.

In LKN, the in-degree $k_{in}$ of a node indicates the number of functions calling it, while the out-degree $k_{out}$ represents the number of functions it calls. Table 1 shows some basic features of the LKN. To comparatively illustrate the features of the LKN, a randomized network with the same number of nodes and edges is constructed. During the process of randomization, self-connections and duplicated edges are avoided. From Table 1, it is clear to see that the LKN consists of 9334 nodes and 26841 edges and it is a sparse network with $k_{in} = k_{out} = 2.8756$. In Table 1, $C$ is the clustering coefficient, which is used to qualify a network’s cluster tendency. It evaluates the extent to which the nodes in the neighborhood of a certain node are connected. Fagiolo [38] extended the clustering coefficient in a directed network as:

$$C = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \left( a_{ij} a_{ik} a_{jk} + a_{ij} a_{jk} a_{ki} \right) a_{ij} a_{ik} a_{jk} - 1) - 2 \sum_{i=1}^{n} a_{ij} a_{ik} \right)$$

where $k_{in}^i$ and $k_{out}^i$ are the in-degree and out-degree of node $i$ respectively, and $a_{ij} = 1$ if there exists an edge from $i$ to $j$, otherwise $a_{ij} = 0$. We can find that the clustering coefficient of LKN is $C = 0.0253$, which is 84 times larger than the value ($C = 0.0003$) of the randomized network. Fig. 2 exhibits the correlation between clustering coefficient $C$ and degrees (both $k_{in}$ and $k_{out}$), we can find that the clustering coefficients are negatively correlated with $k_{in}$ and $k_{out}$. It implies that the neighborhoods of the nodes with small degrees tend to cluster together.

In Table 1, $R$ is a reciprocity parameter, which is used to qualify the symmetry of a network [39]. It can be formulated as follows:

$$R = \frac{\sum_{i<j} (a_{ij} - \bar{a}) (a_{ji} - \bar{a})}{(\sum_{i<j} a_{ij} - \bar{a})^2} \left( \sum_{i<j} a_{ij} - \bar{a} \right) \right)$$

where

$$\bar{a} = \frac{\sum_{i<j} a_{ij}}{n(n-1)}$$

Here, $a_{ij} = 1$ if and only if there exists an edge from $i$ to $j$, otherwise $a_{ij} = 0$. The maximum of $R$ is 1, implying that each edge in the network has a reverse mate. A small $R$ indicates the high asymmetry of the network. Table 1 shows that the reciprocity parameter of LKN is $R = -0.0003$, which is smaller than the value ($R = -0.0001$) of a randomized network. Actually, LKN is an extremely asymmetric network without edge reversing each other, because there is no mutual function call between two functions in the kernel. As a result, it is hard to find a path back in LKN even though there are some loops. In the table, $E$ evaluates the efficiency of information translation [40]. It is defined as

$$E = \frac{1}{n(n-1)} \sum_{i<j} \frac{1}{d_{ij}}$$

Table 1

<table>
<thead>
<tr>
<th></th>
<th>LKN</th>
<th>Randomized network</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>9334</td>
<td>9334</td>
</tr>
<tr>
<td>$m$</td>
<td>26841</td>
<td>26841</td>
</tr>
<tr>
<td>$k_{in,max}$</td>
<td>415</td>
<td>11</td>
</tr>
<tr>
<td>$k_{out,max}$</td>
<td>92</td>
<td>11</td>
</tr>
<tr>
<td>$\langle k_{in} \rangle$</td>
<td>2.8756</td>
<td>2.8756</td>
</tr>
<tr>
<td>$\langle k_{out} \rangle$</td>
<td>2.8756</td>
<td>2.8756</td>
</tr>
<tr>
<td>$C$</td>
<td>0.0253</td>
<td>0.0003</td>
</tr>
<tr>
<td>$R$</td>
<td>-0.0003</td>
<td>-0.0001</td>
</tr>
<tr>
<td>$E$</td>
<td>0.0017</td>
<td>0.1071</td>
</tr>
</tbody>
</table>

¹ Linux is open source software with GNU General Public License which provides users the freedom to run, copy, distribute, study, change and improve the software. Its source codes are available on www.kernel.org.
where, \( d_{ij} \) is the length of shortest path between node \( i \) and \( j \). If there is no path in the network between \( i \) and \( j \), \( d_{ij} = \infty \). We can find that the efficiency of LKN is \( E = 0.0017 \), which is 63 times less than the value \( (E = 0.1071) \) of the randomized network. Obviously, the low efficiency of LKN is caused by its high asymmetry.

Fig. 3(a) and (b) exhibit the in-degree distribution and out-degree distribution, respectively. Degree distribution \( P(k) \) gives the probability of randomly selected nodes having exactly \( k \) degrees. It reflects the basic information about a network’s organization. From Fig. 3, we can find that both the in-degree and out-degree are quite heterogeneous: numerous nodes have small degrees (23.42% nodes with \( k_{in} = 0 \) and 32.12% nodes with \( k_{out} = 0 \)), while only a few of nodes are densely connected (There are only 0.64%nodes with \( k_{in} > 50 \) and 0.02% nodes with \( k_{out} > 50 \)). As shown in Table 1, the maximum in-degree and maximum out-degree in LKN are \( k_{in,max} = 415 \) and \( k_{out,max} = 92 \) respectively. They are much larger than those in a randomized network with same nodes and edges.

Although both the in-degree and out-degree are quite heterogeneous, they follow different distributions. The in-degree follows a power-law distribution while the out-degree follows an exponential distribution. Compared with a randomized network, the maximum of \( k_{in} \) is much greater than the maximum of \( k_{out} \) in LKN. Its out-degree is more homogeneous than in-degree. The difference between the distributions mainly comes from the following reasons:

1. **Small \( k_{out} \)**: The complex functions with too many calls are less reliable and difficult to be maintained. Therefore, in the software development process, developers are less likely to use a complex function, but usually use several simpler functions instead. This principle limits the value of \( k_{out} \).
2. **Large \( k_{in} \)**: In Linux kernel, lots of basic functions (e.g., system status checking) are widely called by various functions, resulting in large \( k_{in} \) values. This is determined by the design principles of operating systems.

Fig. 3(c) shows the relationship between \( k_{out} \) and \( k_{in} \). We can see that \( k_{out} \) and \( k_{in} \) are negatively correlated, especially for large degree nodes. The maximal out-degree of
nodes with \( k_{in} > 30 \) (denoted by red circle) is only 5, while the maximal in-degree of nodes with \( k_{out} > 30 \) (denoted by blue triangle) is only 11. As discussed above, the large in-degree nodes represent the basic service functions and these functions rarely call other functions, resulting in the small value of \( k_{out} \). In contrast, nodes with large \( k_{out} \) are actually related to high level functions, which directly exposed to users, thus they are rarely called by other functions.

### 3. Community characteristics of LKN

For a real networked system, its structure is usually closely related to its functions [41]. Linux is a typical modularized software system, each module of which performs a certain task. In this part, we will identify the modules and further investigate their characteristics.

Community detection is widely adopted to identify the modules of networks. Newman et al. [42] defined modularity to evaluate the quality of a particular division of a network with the expectation that a random network does not have community structure. For a directed network, the modularity can be formulated as follows [43]:

\[
Q = \frac{1}{m} \sum_{vw} \left( A_{vw} - \frac{k_{in}^v k_{out}^w}{m} \right) \delta(C_v, C_w)
\]

where, \( m \) is the total number of edges in the network, \( A_{vw} \) is the adjacent matrix, \( k_{in}^v \) and \( k_{out}^w \) are the node \( v \)'s in-degree and node \( w \)'s out-degree respectively. \( \delta(C_v, C_w) = 1 \) if \( v \) and \( w \) are in the same community (\( C_v = C_w \)). To simplify the definition, let \( e_{ij} \) be the fraction of edges in the network connecting nodes in community \( i \) to those in community \( j \), which can be formulated as:

\[
e_{ij} = \frac{1}{m} \sum_{vw} A_{vw} \delta(C_v, i) \delta(C_w, j)
\]

Thus, \( e_{ii} \) represents the edges that fall in the community \( i \). Let \( \alpha_i^m \) be the fraction of edges in the network that target to the nodes in the community \( i \), and \( \alpha_j^{out} \) be the fraction of edges in the network that originate from the nodes in the community \( j \) respectively. They can be formulated as:

\[
\alpha_i^m = \frac{1}{m} \sum_v k_{in}^v \delta(C_v, i)
\]

\[
\alpha_j^{out} = \frac{1}{m} \sum_w k_{out}^w \delta(C_w, j)
\]

According to these, the modularity can be simplified as:

\[
Q = \sum_i (e_{ii} - \alpha_i^m \alpha_i^{out})
\]

Based on the idea that a well division of a network should have large modularity, Newman proposed a greedy algorithm [44] to maximize the modularity so that one can achieve a suitable division. The inputs of the algorithm are \( n \) communities, each of which only has a single node. The algorithm recursively merges communities to get the maximal increase of \( Q \) at each step. The increase can be calculated as follows:

\[
\Delta Q = e_{ij} + e_{ji} - \left( \alpha_i^m \alpha_j^{out} + \alpha_j^{out} \alpha_i^m \right)
\]

By using sophisticated data structures, Clauset et al. [45] proposed a more efficient community detection algorithm for sparse graphs, called the fast modularity algorithm. In this section, we adopt the algorithm to investigate the module characteristics of LKN.

After executing the algorithm, we can identify 85 communities in LKN. Among them, there are 12 communities with more than 100 nodes, occupying 91.15% nodes of LKN, which we call key communities (Fig. 4). The key communities (C1 to C12) can be classified into 7 groups according to their functions. Table 2 lists the characteristics of the 12 communities. The different color is used to distinguish the 7 groups. We find that there are only two largest communities C1 and C2 with \( l_{in} < l_{in} \), \( l_{out} \) represents the ratio...

---

**Fig. 4.** Connections among 12 key communities. The width of the edge represents the density of the connection between two communities.
Table 2

Key communities of LKN. $l_{in}$ is the ratio of edges target to community $i$ and $l_{out}$ represents the ratio of edges originate from community $i$.

<table>
<thead>
<tr>
<th>Community</th>
<th>Size</th>
<th>$l_{in}$</th>
<th>$l_{out}$</th>
<th>Related function</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>2008</td>
<td>30.44%</td>
<td>23.09%</td>
<td>Basic services</td>
</tr>
<tr>
<td>C2</td>
<td>1817</td>
<td>28.01%</td>
<td>19.82%</td>
<td>Process scheduler</td>
</tr>
<tr>
<td>C3</td>
<td>1636</td>
<td>16.91%</td>
<td>19.23%</td>
<td>Trace system</td>
</tr>
<tr>
<td>C4</td>
<td>449</td>
<td>2.70%</td>
<td>5.15%</td>
<td>Runtime locking correctness validator</td>
</tr>
<tr>
<td>C5</td>
<td>950</td>
<td>6.87%</td>
<td>8.89%</td>
<td>Insert mechanism</td>
</tr>
<tr>
<td>C6</td>
<td>644</td>
<td>3.40%</td>
<td>3.71%</td>
<td>Basic services</td>
</tr>
<tr>
<td>C7</td>
<td>304</td>
<td>2.01%</td>
<td>2.79%</td>
<td>Device management</td>
</tr>
<tr>
<td>C8</td>
<td>550</td>
<td>0.74%</td>
<td>2.12%</td>
<td>Time functions</td>
</tr>
<tr>
<td>C9</td>
<td>220</td>
<td>2.25%</td>
<td>2.49%</td>
<td>Interrupt mechanism</td>
</tr>
<tr>
<td>C10</td>
<td>115</td>
<td>0.88%</td>
<td>1.16%</td>
<td>Time functions</td>
</tr>
<tr>
<td>C11</td>
<td>154</td>
<td>1.00%</td>
<td>2.28%</td>
<td>Interrupt mechanism</td>
</tr>
<tr>
<td>C12</td>
<td>115</td>
<td>1.13%</td>
<td>2.76%</td>
<td>Device management</td>
</tr>
</tbody>
</table>

of edges originate from community $i$ and $l_{in}$ is the ratio of edges target to community $i$. They are formulated as:

$$l_{out} = \frac{\sum_{i \neq j} e_{ij}}{\sum_{j \neq i} e_{ij}} \tag{11}$$

$$l_{in} = \frac{\sum_{i \neq j} e_{ij}}{\sum_{j \neq i} e_{ij}} \tag{12}$$

The functions in a community with $l_{out} < l_{in}$ are more likely to be called in the system than to call other functions. By analyzing source codes, we find that most functions in C1 and C2 provide basic services widely used in the kernel. So we can infer that C1 and C2 are the foundation of the LKN. The functions in C3 and C5 perform the main task of the kernel which is to schedule the processes. The functions in C4, C6, C7 and C10 are assistant functions to track system status. With the help of the useful information, developers can trace the system and find out the system defects conveniently and efficiently. The functions in C8, C9, C11 and C12 are also necessary for the operation of the kernel. C8 refers to the runtime locking correctness validator which is used to guarantee the security of system resources. C9 is related to the time functions and C11 manages system devices. C12 refers to the interrupt mechanism which is used to communicate with the devices.

4. Robustness of LKN

In this section, we will investigate the robustness of LKN according to the mechanism of software failures. Robustness of a network refers to the ability to avoid crash when a fraction of its nodes or edges fail. It is an important property of many real and complicate systems such as Internet [46] and power grids [47].

For a complex software system, Voas [48] proposed the PIE model to examine the mechanism of software failures. When a program fails, it is certain to say that at least one fault exists. However, the existence of a fault does not guarantee the failure of the program. To turn a fault into an observable failure, firstly the faulty statements must be executed. After that, the system should show some unexpected behaviors. Furthermore, the abnormal states need to be propagated in the system till the users notice them. Considering these, the Execution of faults, Infection of system and Propagation of abnormal states are necessary conditions to cause the failure of a program. For $i$-th function in a program, let $P_i$ denote the propagation probability of its abnormal states, $I_i$ the probability that it is infected and $E_i$ the probability that at least one fault in the function has been executed. Thus, we have:

$$P_i = pl_i \tag{13}$$

$$I_i = E_i + \bar{E_i} \left(1 - \prod_{j=1}^{n} a_{ij} P_j \right) \tag{14}$$

$$\bar{P_i} = 1 - P_i \tag{15}$$

$$E_i = 1 - E_i \tag{16}$$

where $p \in [0, 1]$ is a parameter to specify the propagation probability of abnormal states, and $a_{ij} = 1$ if and only if function $j$ is called by $i$ (otherwise $a_{ij} = 0$). Obviously, once a node of LKN fails, it may lead to cascading effect. Fig. 5 shows a mini illustration of the process. At first, the system works well (Fig. 5 (a)). Then node $f$ failed due to error or attack factor (Fig. 5 (b)), resulting in the cascading failure of node $e$ and $d$ (Fig. 5 (c)). Finally, node $a$ is infected (Fig. 5 (d)).

![Fig. 5. An illustration of function failure propagation. (a): original network. (b) Some faults in the function $f$ were executed. (c) The abnormal behaviors of $f$ are propagated with probability $p$. $e$ and $d$ are infected (d) function $a$ is infected further.](image-url)
The infection probability $I$ is the evaluation standard of system reliability. The larger the value of $I$ is, the higher risk the system will have. From Fig. 6(a), we can see that the value of $I$ grows as the initial fraction of failure nodes $f$ increases under the three failure strategies: random failure, $k_{out}$ and $k_{in}$ based intentional attacks. Among the three cases, the $k_{in}$ based intentional attack brings the network highest risk. As discussed above, the nodes with large $k_{in}$ usually relate to the functions providing basic services. The failure of these nodes will induce more severe propagation in the system. On the contrary, the nodes with large $k_{out}$ usually relate to high level functions which are rarely called, and thus the $k_{out}$ based intentional attack can hardly induce cascading failures. Fig. 6(b) shows the infection probability $I$ as the function of $p$. It is clear that the value of $I$ also monotonously increases with the increment of $p$ for all the three cases. Besides, the nodes are most likely to fail under the $k_{in}$ based intentional attack, while they are least likely to fail under the $k_{out}$ based intentional attack. From the discussion of $f$ and $p$, we can infer that the faults in basic functions can bring the whole system high risk to fail.

We further investigate the network robustness of LKN. Fig. 6(c) shows the relative size $G$ as the function of $f$. We can find that the network is robust against random failure, but is vulnerable when facing intentional attacks. Specifically, the network soon breakdown as $f$ increase under the $k_{in}$ based intentional attack. Fig. 6(d) shows the relationship of $G$ and $p$. It is obvious that the network becomes more and more vulnerable as $p$ increase. Again, the network is most fragile when facing the $k_{in}$ based intentional attack. These phenomena indicate that the network is extremely vulnerable under the $k_{in}$ based intentional attack.

5. Conclusion

In this paper, we study the Linux kernel from the network point of view to investigate the underlying principles of operating system. It is found that the LKN’s out-degree is more homogeneous than its in-degree. Community detection shows that LKN is a highly modularized network with 12 key communities. They can be classified into 7 groups, and each group refers to a particular mechanism of the Linux kernel. Moreover, the robustness of LKN is examined according to PIE model. We find that the failure of the nodes with large in-degree do more damage on the whole system. This work is helpful in understanding complex software systems.

Acknowledgements

The authors would like to thank Dr. Wen-Bo Du from Beihang University for kindly providing valuable technical advices to this manuscript.

References