Evolution of Function-Call Network Reliability in Android Operating System

Anzhuo Yao, Pengfei Sun, Shunkun Yang, *Member, IEEE*, and Daqing Li, *Member, IEEE*

***Abstract*— Operating systems (OS) are critical infrastructures for information system. To design a highly reliable software, it is essential to understand the architecture feature of operating systems, which is recently explored by network analysis. While most focus is on the topological properties, the network reliability is rarely studied. In this paper, based on percolation method, we analyze the function-call graph of Android OS in different levels. While OS network is more vulnerable under degree-based percolation at node level, it becomes more vulnerable under strength-based percolation at community level. Furthermore, we found that although topological properties of kernel network are evolving with different released versions, percolation properties seem rather stable. Our findings may help to understand the reliability principle of OS architecture and to design new system testing methods.**

***Index Terms*— Complex networks, evolution, Android OS kernel, percolation, reliability.**

I. Introduction

AS ONE of the largest man-made systems, the software system is a typical complex system and will become more complicated due to the increasing and multiple requirements. For instance, even a simple component of a software system needs more than millions of LOC (lines of code). Thus, it is urgent to well understand the interaction in this complex system, given the frequent failures even if it is written by experts [1]. Different from hardware systems which require reliable circuit design [2], [3], the traditional software analysis methods have mainly focused on the code level. Complex network theory, as an emerging and hot analytical tool for complex systems, can be used to analyze the complex software system in the architecture level which also supports network reliability analysis using both qualitative and quantitative methods. The application of complex network theory to the software system emerged recently by Myers CR [4]. Concas *et al.* [5] and Louridas *et al.* [6] subsequently observed the power-laws in software system. The scale-free and small-world features were further identified in software networks by Valverde *et al.* [7]. While the complex network theory has been applied to software system for an in-depth understanding of its structure characteristic, the reliability feature of corresponding software network and its evolution remains unclear. The research of network reliability focuses on the ability of system to withstand various perturbations. Classical network reliability studies terminal reliability with probability calculations [8], [9], mostly for hardware system. For software system, network reliability is also essential yet unsolved for its critical role in the system design and operation.

Therefore, percolation theory, as an emerging approach [10], is utilized to identify the operational limit and study how this limit evolves with updates of complex system. In this paper, we investigate the reliability evolution of Android kernel by modeling a function-call network, whose nodes and edges are respectively functions and call relations between them extracted from kernel's source code. The main contributions of this paper can be listed as follows: 1) the community analysis to abstract the network and establishment of the community -interaction network revealing the function interaction in the large scale; 2) the reliability of kernel network is analyzed by percolation theory in different scales; 3) the topological evolution of function network from different kernel versions; 4) reliability evolution of function-call network across different kernel versions.

The organization of the paper is as follows. The related work is arranged in Section II. Section III introduces the preliminaries. In Section IV, analysis of single function-call network is depicted in three aspects. Section V analyzes evolution of function-call network in different release versions. Section VI draws conclusions.

II. Related Work

Nowadays, the demand of smartphones is fast growing, which requires to study the reliability of their operating systems as foundation of the whole software systems in smartphones. As one of the main operating systems, reliability of Android system is critical for the system performance, especially with various interactions among a huge number of functions. Android is a Linux-based operating system (OS) with its framework consisting of 5 levels from bottom to top: Linux Kernel, HAL, Native Libraries & Android Runtime, Android Framework and Applications. The major faults in Android OS occur in the following components [11]: Audio, Camera, Bluetooth, Phone, Kernel, WIFI, etc. Due to its

Manuscript received May 14, 2019; revised October 14, 2019 and December 27, 2019; accepted February 5, 2020. Date of publication February 14, 2020; date of current version April 1, 2020. This work was supported by the National Natural Science Foundation of China under Grant 71822101. This article was recommended by Associate Editor W. X. Zheng. *(Corresponding author: Daqing Li.)*

The authors are with the School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China (e-mail: yaodszx@163.com; sun_pengfei@foxmail.com; ysk@buaa.edu.cn; daqingl@buaa.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2020.2972995

essential role in system function, the failures in kernel may lead to the fatal error in the whole system, so that the analysis of kernel is essential and valuable for the reliability and stability of Android OS. C. Wright *et al.* [12] proposed a general security support method to minimize the impact to Linux kernel by using LSM (Linux Security Module). Gu *et al.* [13] observed the Linux kernel behavior under errors by experiments.

Over the past few decades, it has become more and more universal to analyze real-world complex systems from a complex network perspective, through modeling the entities and their interactions as nodes and edges of networks. Due to the limited model ability of regular network, Erdös and Rényi proposed Erdös-Rényi (ER) random network model [14] in1959. However, while neither regular lattices nor random networks can describe real-world complex systems, there are two famous network models introduced at the end of last century: the small-world network model [15] and the scale-free network model [16]. After that, with the deeper understanding and blossoming of complex network science, researches have been greatly prompted different network models in diverse domains, such as World Wide Web [17], cyber-physical systems [18], [19], epidemic spreading [20], [21], traffic dynamics [22]–[24], network control [25], [26], power grids [27]–[30], airport networks [31] etc. Given the complexity of modern software system, one attractive aspect is to use complex network model to understand software systems.

With complex network theory, the investigation of software system is based on the exploration of components and their dependencies called components dependency network (CDN) [32]. Researchers make contribution to the model and topological analysis of software network in different granularities: packages, class, method or function level and multi-granularity with the combination of the three mentioned above [32]–[34]. The structure of software network has been studied with the utilization of community analysis combined with functionality in different software and multiple versions of the same software [35], [36]. The software execution processes have been first modeled as evolving complex network [1] and been promoted by analyzing the key node behavior in software execution network [37]. Except for the research mainly focusing on some specific software, the study of OS in complex network perspective has also been performed recently. Yan *et al.* [38] analyzed the topology and evolution of Linux-kernel network compared with genomes by building the hierarchical structures. Two new network growth models were developed by Zheng [39] to analyze Gentoo Linux whose network cannot be easily explained by existing complex network models. The investigation of Gao *et al.* [40] shows that critical nodes in kernel's complex network tend to have large in-degree by conducting percolation analysis to the whole network. The coupling relationships between Linux's components were uncovered by Wang *et al.* [41]. They also compared networks extracted from normal and failure status of Linux OS to perform effects analysis of system failures on networks. Xiao *et al.* [42] studied the evolution of 62 major releases of the Linux kernel by utilizing complex network theory and analyzed changing events among versions in network perspective.

III. Preliminaries

In this section, some preliminaries about the model and analysis methods are briefly introduced.

A. Research Data

Android OS is based on Linux kernel, which has more than 1300 releases ranging from version 1.0 to 5.1 (the latest release version) over the past 20 years. Since the main released versions utilized in Android OS are a series of 4.x versions such as 4.4, 4.9, 4.14 and 4.19 released version, the research data of function-call network are obtained from 21 releases ranging from version 4.0 to 4.20.

B. Network Modeling

The kernel of Android OS contains a great deal of functions which are capable of calling each other. Therefore, the modeling of the function-call network is an effective approach to visualize the interaction between these functions in the kernel. In order to extract data of function call, we modify the kernel's makefile and then use the regular expression module in python to calculate the nodes and edges. Finally, we generate the function-call network model in kernel from the compiling process of kernel source code. In addition, since the actual utilization of Linux kernel never loads all existing modules, a config based on the fundamental arm64 CPU architecture is implemented during the compiling process and the kernel studied here will only contain modules in that config with the file location——'/arch/arm64/config/defconfig'.

Obviously, the model is a directed network. There is a demo of function-call network and the source code shown in Fig. 1, which is generated by a part of C file located at '/arch/arm64/kernel/module.c'. Fig. 2 (for better visualization, the edges have been removed) depicts the whole function-call network of kernel V4.9 using different colors to identify different models, such as drivers, kernel, fs, mm, etc.

The function-call network can be defined as a directed network $G\ (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of N nodes in accordance with functions. The interaction among each couple of function vertexes v_s and $v_t (v_s, v_t \in V)$ is represented by $E = \{e_1, e_2, \ldots, e_M\}$ which is the set of M edges with every single $e_i = (v_s, v_t)$ $(i = 1, 2, \ldots, M)$. All network statistics used in this paper are presented as follows.

1) Average Degree $k_{average}$: In-degree k^{in} implies the number of a given function called by other functions, and out-degree k^{out} symbolizes the number of functions called by a given function. The total degree k of a directed network is the sum of k^{in} and k^{out}. The $k_{average}$ in this paper represents the average total degree k as follows:

$$k_{average} = \frac{\sum_{N} (k_i^{in} + k_i^{out})}{N} \tag{1}$$

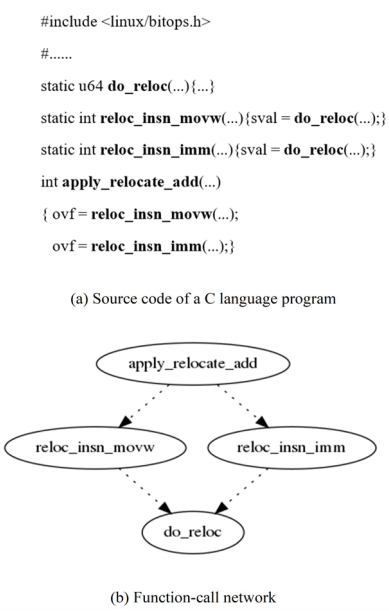


Fig. 1. A display for transforming function-calls into a directed network. (a) A fraction of source code in Linux kernel with simplification. (b) The corresponding function-call network whose nodes and edges represent functions and call relations extracted from (a).

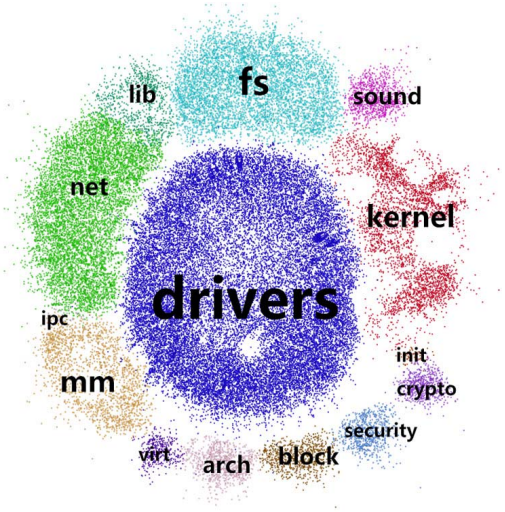


Fig. 2. Function-call network. This is a diagrammatic drawing for the stable version 4.9 of Linux kernel in a network perspective. Each node represents a function in kernel and edges have been removed for a better visualization. Different colors of network are utilized for labeling different modules.

2) Degree Distribution P(k): The degree distribution is an effective method to identify the category of network models, and reflects the topological structure of software system. It suggests that the degree in this function-call network seems a power-law distribution as follows:

$$P(k) \propto k^{-\gamma} \tag{2}$$

where $P(k)$ is the probability of nodes in the network having degree k, and γ is the scaling parameter.

3) Betweenness: Betweenness is an important indicator [43] for metric of node significance in the whole network integration. The betweenness of a node v_i in a network is the ratio between $S_{jl}(i)$, the number of the shortest paths between any of two nodes v_j, v_l passing through node v_i, and S_{jl}, the total number of shortest paths. Here we assume that network has N nodes. The expression appears as shown:

$$B_i = \sum_{\substack{j,l \in N \\ j \neq l \neq i}} \left[\frac{S_{jl}(i)}{S_{jl}} \right] \tag{3}$$

4) Clustering Coefficient C: The probability that one vertex's neighbors in a network are linked with each other is utilized to define the clustering coefficient. It reflects the local aggregation of network. The tightly connected neighborhoods will get larger clustering coefficient. As an unweighted network is determined, the C of vertex i is defined as [44]:

$$C_i = \frac{\frac{1}{2} \sum_{j \neq i} \sum_{h \neq (i,j)} a_{ij} a_{ih} a_{jh}}{\frac{1}{2} k_i (k_i - 1)} \tag{4}$$

where $a_{ij} = 1$ (the basic component of adjacency matrix) when there is a link from vertex i to j; if not, $a_{ij} = 0$. Item $a_{ij} a_{ih} a_{jh}$ represents the existence of a triangle around i. For the entire network, the clustering coefficient C is depicted as:

$$C = \frac{1}{N} \sum_{i=1}^{N} C_i \tag{5}$$

5) Average Shortest Path Length d: The average shortest path length equals to the mean of all shortest steps between two vertexes in a network. It can represent network's efficiency of information or mass transport. The average shortest path length is denoted as follows:

$$d = \frac{1}{N(N-1)} \sum_{v_s, v_t \in V} d(v_s, v_t) \tag{6}$$

where $d(v_s, v_t)$ is the shortest path length from s to t.

C. K-Core Analysis

When a given network G (V, E) is determined, a k-core [45] is equivalent to a subnetwork $G_h = (V, E|V_h)$ composed of the set V_h belonging to V if and only if $\forall v_i \in V_h : k_i \geq k$. Remind that G_h is the maximal subnetwork with this statistic.

1) Coreness: The vertex in the k-core but not in the $(k+1)$-core is identical to the node which has a coreness of k. The maximal coreness $k_{\max}$, the same to graph coreness, is considered as the main core of a graph. Notice that the $(k_{\max}+1)$-core is empty, but the $k_{\max}$-core is not.

2) Core Size: The number of vertices in k-core is utilized to represent core size.

3) K-Shell: All vertexes with coreness k form a k-shell S_k. Therefore, the k-core consists of all S_q with $q \geq k$.

As is depicted in Fig. 3, there is a display of k-core decomposition where the peripheral dotted line contains 1-core. 2-core will be apparent after removing all of blue nodes whose degree equals to 1. And the innermost part whose nodes are red is the 3-core part, the main core of this graph.

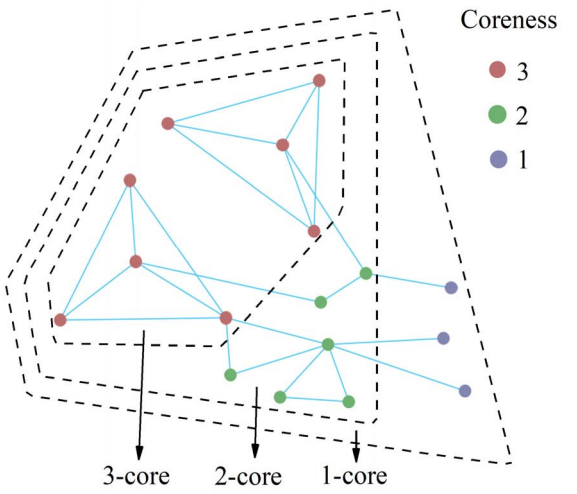


Fig. 3. An example of k-core decompostion. The coreness of this network k_{max} is 3. We use three dotted line to divide the network into three parts. The red part is 3-core, also 3-shell, with nodes whose degree equals to 3 after removing other nodes. The middle part with green color is 2-shell and the both red and green part constitute 2-core. The peripheral part in network is 1-shell with blue color.

IV. Analysis of a Single Function-Call Network

In this part, the topological feature of function-call network has been revealed. We then study the community structure and the percolation analysis is implemented.

A. Degree and Betweenness Distribution of Function-Call Network

The function-call network of kernel V.4.9 is analyzed in this section for studying the distribution of both degree and betweenness.

The log-log plots of in-degree and out-degree distribution are shown respectively in Fig. 4. The result of fitting shows that the γ_{in} is around 2.409 and γ_{out} is around 3.554. We can find that the distributions of k_{in} and k_{out} signify the heterogeneity between different functions (k_{out} of 13.41% nodes equals to 0, and of 1.06% nodes equals to 10; k_{in} of 36.11% nodes equals to 0, and of 0.35% nodes equals to 10). The large amount of nodes with small k_{in} and k_{out} also implies that most of nodes are the starting or ending functions in a process, which initiate a function call or terminate a call-trace respectively. For the gap between $k_{in-\max}$ and $k_{out-\max}$, it is suggested [38] that programmers always tend to abandon the usage of a function with too many calls for the enhance of reliability, and some fundamental functions are frequently called by multiple functions.

The Fig. 5 shows that the betweenness of function-call networks also seems to follow a power-law distribution (γ_B is around 1.854). This is related to the heterogeneous distribution of degree, which cause a heterogeneous distribution of shortest path. Meanwhile, since the software system usually has a modular structure, nodes connecting different modules also have a high betweenness. In the following we will further study the community structure of kernel system.

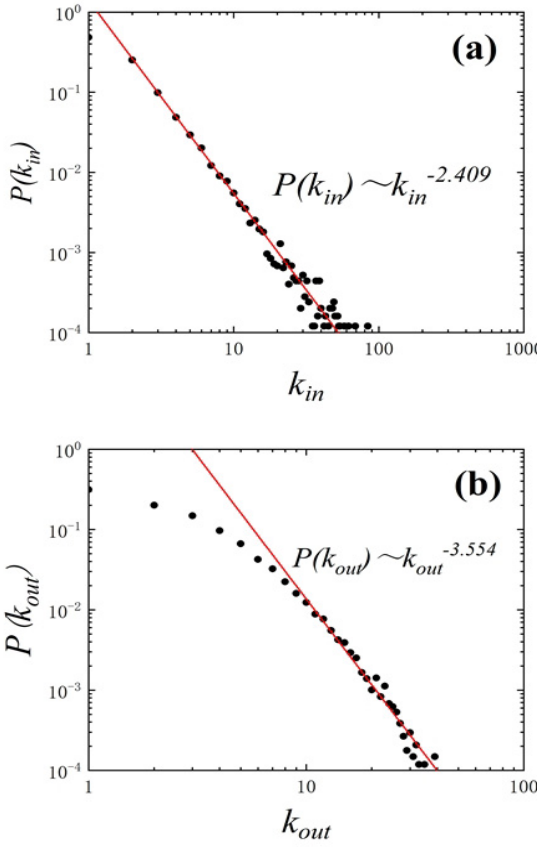


Fig. 4. Degree distribution of function-call network of Linux kernel V4.9. (a) In-degree distributions (log-log plot) in function-call network. (b) Out-degree distributions (log-log plot) in function-call network. Both k_{in} and k_{out} seem to follow a power-law distribution with $\gamma = 2.409$ and 3.554 respectively.

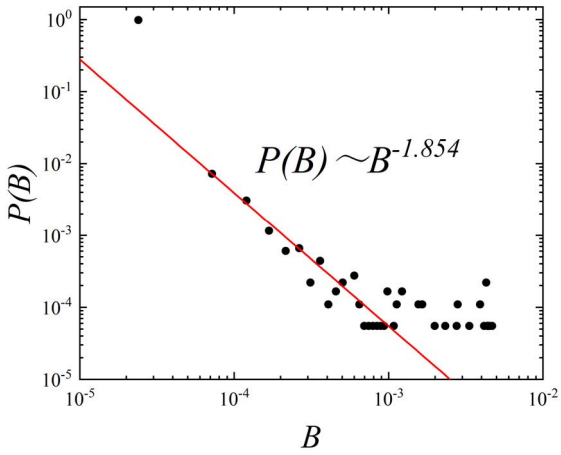


Fig. 5. Betweenness distribution (log-log plot) in function-call network of Linux kernel V4.9. The betweenness of function-call networks seems to follow a power-law distribution (γ_B is around 1.854).

B. Community Structure Analysis

In this subsection, we implement community analysis in function-call network based on the kernel V4.9 and make an effort to reconstruct the network in a community perspective.

1) The Community Structure of Kernel: As a typical software system, kernel of Android OS could have a community

structure. There are two main approaches to analyze the community structure: one is based on code repositories and the other is based on topology analysis. The community structure analysis based on code repositories is generated by developers. For instance, there are several folders in directory of kernel source code: arch, drivers, documentation, init, kernel and so on, while different folders represent different features. This approach to analyze community structure is convenient for operation. Meanwhile, classification by developers may not reflect essence of interaction between communities if the code repositories designed improperly. The community structure analysis based on topology analysis is more complicated, however, it can steadily help researchers comprehend the community structure. We use Louvain algorithm [46] based on modularity to analyze the community structure of android OS kernel.

Focus on the mesoscopic level of network structure, a network can be divided into different clusters by utilizing the modularity as a metric [47]. Modularity is defined as follows:

$$\begin{cases} Q = \dfrac{1}{2M}\sum_{i,j}\left[A_{ij} - \dfrac{k_i k_j}{2M}\right]\delta\left(c_i, c_j\right) \\ \delta\left(u, v\right) = \begin{cases} 1 & u = v \\ 0 & u \neq v \end{cases} \\ k_i = \sum_j A_{ij} \\ M = \dfrac{1}{2}\sum_{ij} A_{ij} \end{cases} \tag{7}$$

where A_{ij} is the adjacency matrix of the network, k_i is the degree of v_i, c_i is the community v_i belongs to, and M is the total number of edges. According to this algorithm, we eventually reveal the kernel's community structure with 242 components to build a community-interaction network in the following.

2) Community-Interaction Network Analysis: To analyze the interaction of the communities in kernel, we model the network of community-interaction, whose nodes and edges are communities and interactions between communities respectively. In the function-call network, 457 nodes which can form 162 communities are not connected to maximum connected subgroup, i.e. the communities built by those 457 nodes cannot interact to the communities in maximum connected subgroup so that we only analyze 80 communities which have interactions. The weights of edges are the amounts of the call times between two communities. We finally get a directed-weighted community-interaction network which can be visualized in Fig. 6. The diameter of a node reflects the quantity of functions contained in this community. The nodes of darker color have a higher degree than those of light color. In addition, the weight of the edges is measured by the number of function-calls between different communities which is in proportion to the width of links. The network features are shown in TABLE I. Strength S_i is an indicator to measure the information-exchange strength of the node v_i [48], and is

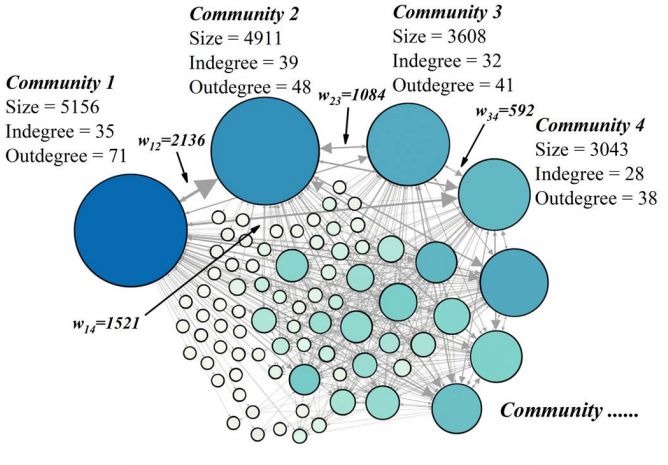


Fig. 6. Community-interaction network. The nodes and edges of community -interaction network are communities and their interactions respectively. The size of a node community indicates the number of functions belonging to this community. The weights of edges (w_{ij}) are the amounts of the call times between two communities. The shade of color is utilized to signify the degree of node which will be darker with higher degree.

TABLE I
STATISTIC FEATURES OF COMMUNITY-INTERACTION NETWORK

Symbol	Statistic Feature	Value
N	number of nodes	80
M	number of edges	768
$k_{in\text{-}average}$	average in-degree	9.6
$k_{out\text{-}average}$	average out-degree	9.6
$k_{in\text{-}max}$	maximal in-degree	39
$k_{out\text{-}max}$	maximal out-degree	71
$B_{average}$	average betweenness	0.00799
B_{max}	maximal betweenness	0.2487
$S_{average}$	average strength	881.975
S_{max}	maximal strength	11647

defined as follows:

$$s_i = \sum_{j \in N} w_{ij} \tag{8}$$

where w_{ij} is the weight between node v_i and node v_j

We find that there are relationships between the size and the topological features for a given community. As is demonstrated in Fig. 7, we normalize the degree, strength and betweenness of each community by its maximum value. The relation between strength and size n is approximately linear, suggesting that large community takes more responsibility for information -exchange task of the whole system. While degree is increasing with community size in a log function, betweenness is likely to be a quadratic function of size. This suggests that the integration of the whole system is positively correlated with node degree, leading to a fast growth of community betweenness with community size. In this way, we can predict the topological statistics for a given community. The result can be conductive to the design of kernel's functions.

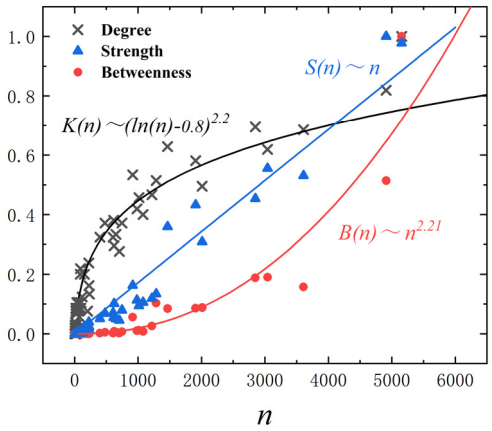


Fig. 7. Relationships between the size of community n and three topological features (degree, strength and betweenness) in community-interaction network. For a better comparison result, three topological features are normalized by its maximum value. The black curve shows degree is increasing with community size in a log function, while betweenness is likely to be a quadratic function of size labeled by red. The blue line indicates that relation between size n and strength is approximately linear.

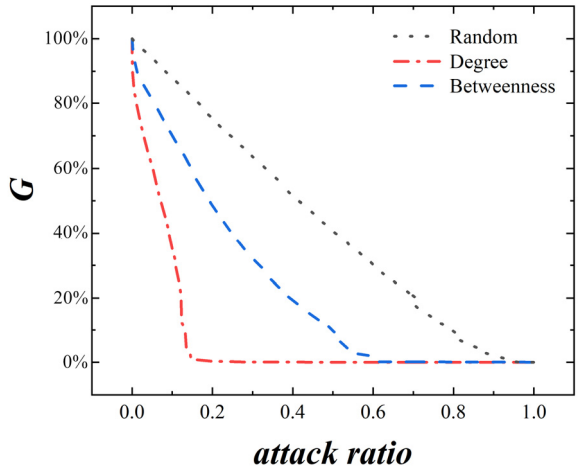


Fig. 8. Percolation process in function-call network. G is the proportion of maximal connected component in the whole network and attack ratio is the percentage of the nodes we remove in the network. The percolation process contains three node attack modes: random (black dotted line), degree-based (red dotted dash line) and betweenness-based (blue dash line).

C. Percolation Analysis

Percolation theory is a method to analyze the reliability of networks when the networks are perturbed or attacked [10]. With the increasing removal ratio of nodes or links, the critical structures can be exposed in percolation process. We can measure the size of maximal connected component and observe the variation of its size with removal ratio. In this way, we can analyze the robustness of the system and find out critical nodes with special topological features.

1) Percolation Analysis of Function-Call Network: We attack the function-call network by following modes: random, based on degree or based on betweenness. The result of the attack is shown in Fig. 8. In Fig.8, G is the size of maximal connected component and attack ratio is the percentage of the nodes we remove in the network. We found that the function-call network has a relative good robustness under random attacks, while it become vulnerable if we remove nodes according to its degree. We can identify the nodes with the higher degree as the critical functions in the function-call network such as the "kfree" (with its degree equals to 3530), a function that release the memory in a dynamic way, and "printk" (with its degree equals to 3485) which is a basic function to control the print of output.

2) Percolation Analysis of Community-Interaction Network: Instead of node level, percolation of community level can tell more information in the large scale. Based on the community analysis in Section 3, the community-interaction network is attacked by the same three modes above, together with the strength-based mode. The result of the attack is shown in Fig.9. In Fig. 9, the diagram suggests that the velocity of network breakdown is highest under the mode of strength-based attack, while the network performance of the degree-based is almost identical to that of betweenness-based attack. Fig. 9 also illustrates that the critical nodes are the communities with higher strength in the community-interaction network. For example, the community with highest strength includes 5156 functions.

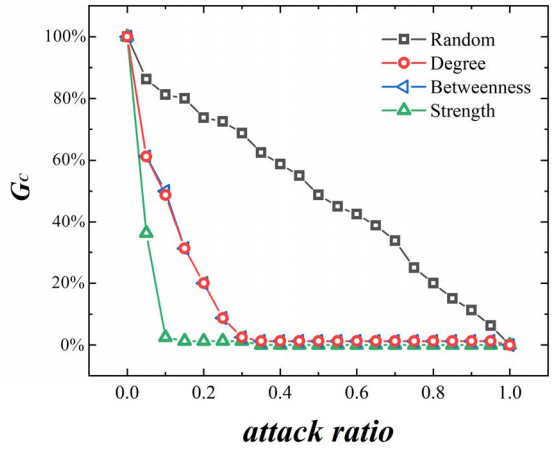


Fig. 9. Percolation process in community-interaction network. G_c is the proportion of maximal connected component in community-interaction network and attack ratio is the percentage of the nodes we remove also at community level. The percolation process contains four node attack modes: random (black squares), degree-based (red circles), betweenness-based (blue triangles) and strength-based (green triangles).

To better understand and visualize the process of percolation, Fig. 10 is assigned to monitor the percolation under degree-based attack mode by showing the community-network with three different attack ratio. The network covered by red is maximal connected component and the grey parts depict failure. Under each degree-based attack ratio, the different organization scale of the whole software network is emerged gradually.

V. Evolution of Function-Call Network

A. Evolutions of Topological Properties

In this part, analysis of the evolutions of network properties over 21 major Linux kernel releases is proposed.

According to the information provided by Linux official [49], the released time of stable kernel ranging from

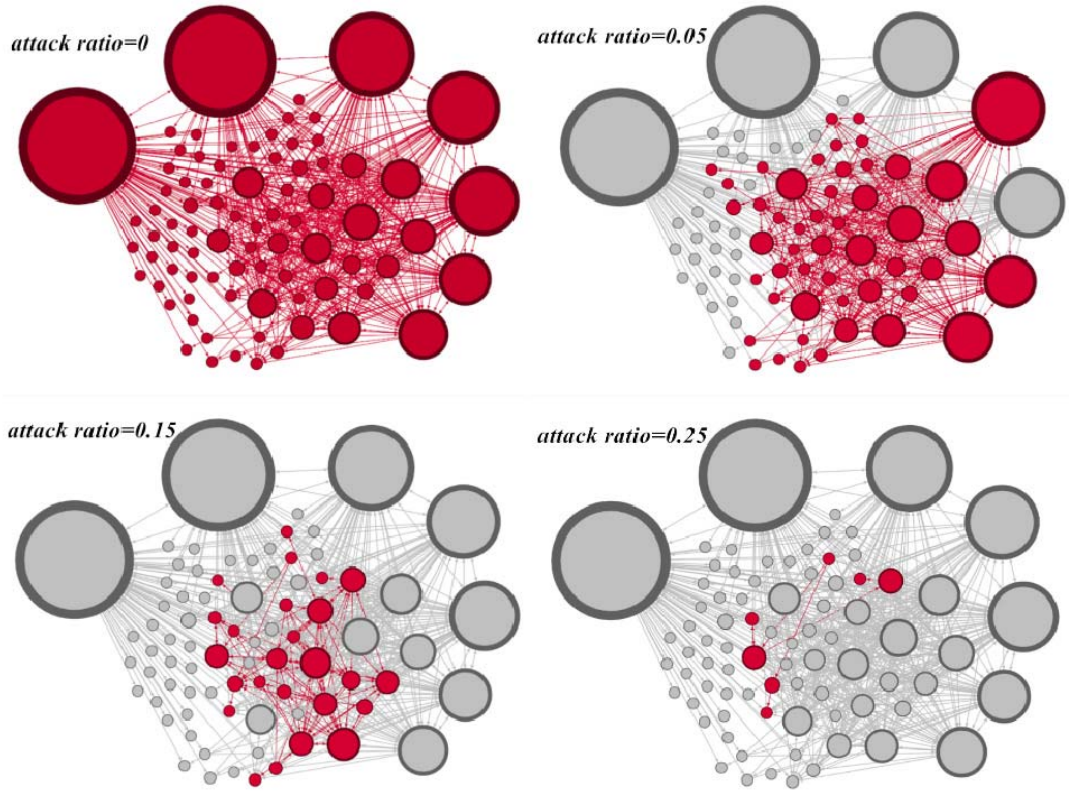


Fig. 10. The visualization of percolation process in community-interaction network. Four snapshots have been chosen in the degree-based percolation when the attack ratio equals to 0, 0.05, 0.15 and 0.25 respectively. The normal nodes and their edges are shown in red and become grey when they have been attacked.

V4.0 to V4.20 in sequence lasted from April 12 in 2015 to December 23, 2018, during which the growth of software development is both rapid and changeable, under the urge requirement of the updated hardware. This trend becomes obvious and inevitable in kernel of Android OS, a highly complex and fundamental software level as well. As is depicted in Fig. 11(a), the scale of kernel network increases almost linearly with the specific version respectively, after the abrupt increase at version 5. This is also confirmed by the increase of total number of network edges in Fig. 11(b), suggesting the relation between network size and its interactions.

The development of software can be classified into 2 major actions: adding new functions, and adding new calling relations. We can observe the competition between these 2 actions from network average degree. For the evolutions of the average degree $k_{average}$ in Fig. 11(c), we select a few specific versions whose $k_{average}$ tend to greatly change compared to the previous release to divide the whole trend into three stages. With a sharp descending at the end of first stage at version 3, the $k_{average}$ of kernel network decreases to the minimum at version 5 which may be caused by adding more basic functions with low interactions. In next stage from V4.8 to V4.15, the degree k seems rather stable. It is suggested that some internal balance is kept when adding multiple new features to satisfy the requirement of users. After V4.16, $k_{average}$ begins to increase. The trend of $k_{average}$ is similar to that of the clustering coefficient C in Fig. 11(d). This suggests the updated function networks are mostly localized and influence the clustering coefficient in the same way. We will check this deeply in the following k-core decomposition analysis.

Fig. 11(e) displays the variation of betweenness B with the increasing of sub-version sequence. Because of the modularity and decoupling features of Android OS, it turns out to be heterogeneous distribution of betweenness of kernel network. After V4.3, average betweenness is decreasing with a sharp decline until version 8. Adding new functions will increase the total information load, yet generate more new paths to share the load between the same pair. This may suggest that the software network is becoming more balanced. The tendency of the average shortest path length d is described in Fig. 11(f). Shortest path length represents the short execution path in kernel of Android OS. Path length is increasing in the early version due to adding new functions. When reaching maximum, path length begins to decrease due to adding more new interactions between functions.

B. Evolutions of k-Core Structures

As is demonstrated in Fig. 12, the maximal coreness of network is increasing with growth of version number. This result depicts that the structure of kernel network becomes more complex with more hierarchies. Adding new functions in the periphery of original network can increase the graph coreness. It is checked in Fig. 13 about the correlation

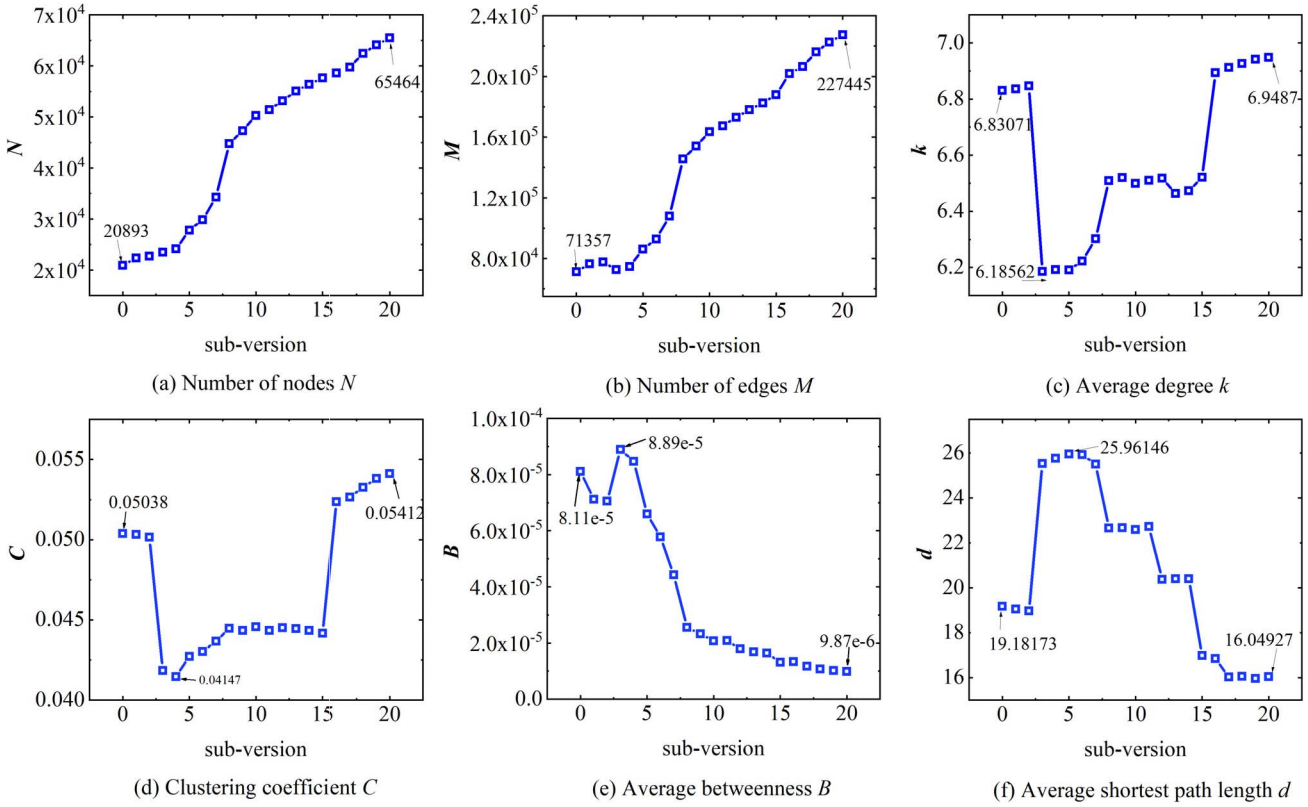


Fig. 11. Evolutions of topological properties of function-call network. (a and b) The evolution of network scale in all Linux kernel versions from V.4.0 to V.4.20. There are four basic topological features chosen to make further evolution analysis: Average degree (c), Clustering coefficient (d), Average betweenness (e) and Average shortest path length (f).

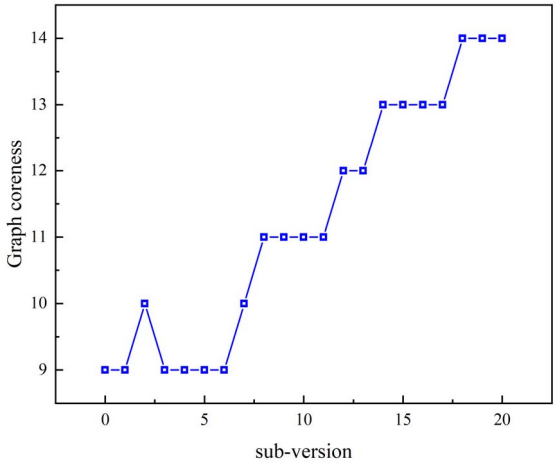


Fig. 12. Evolution of graph coreness in function-call network of 21 kernel releases. Graph coreness seems to grow step by step with the version updates of Linux kernel.

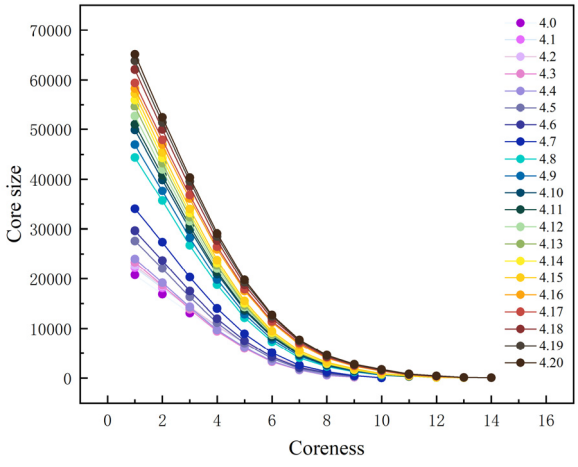


Fig. 13. Relations between core size and coreness in different kernel version. Different colors are utilized to distinguish the versions from V4.0 to V4.20. In each curve, core size is decreasing with increasing coreness.

between the core size in different kernel versions and the corresponding coreness. For a given version, core size is decreasing with increasing coreness, suggesting a tree-like structure for function-call network. With the evolution of kernel version, the size of each core increases for a given coreness, where the size difference between different shells is the most at periphery. The core size of the central part of network is rather small, which seems performing different basic functions. Furthermore, the speed of decreasing along the coreness becomes faster with the version iteration. The result shows that most of new functions are added at the periphery of the kernel structure.

For a better understanding and visualization of the evolutions of k-core structures, Fig. 14 displays the 9-cores of kernel networks in some specific versions containing V4.3, V4.7,

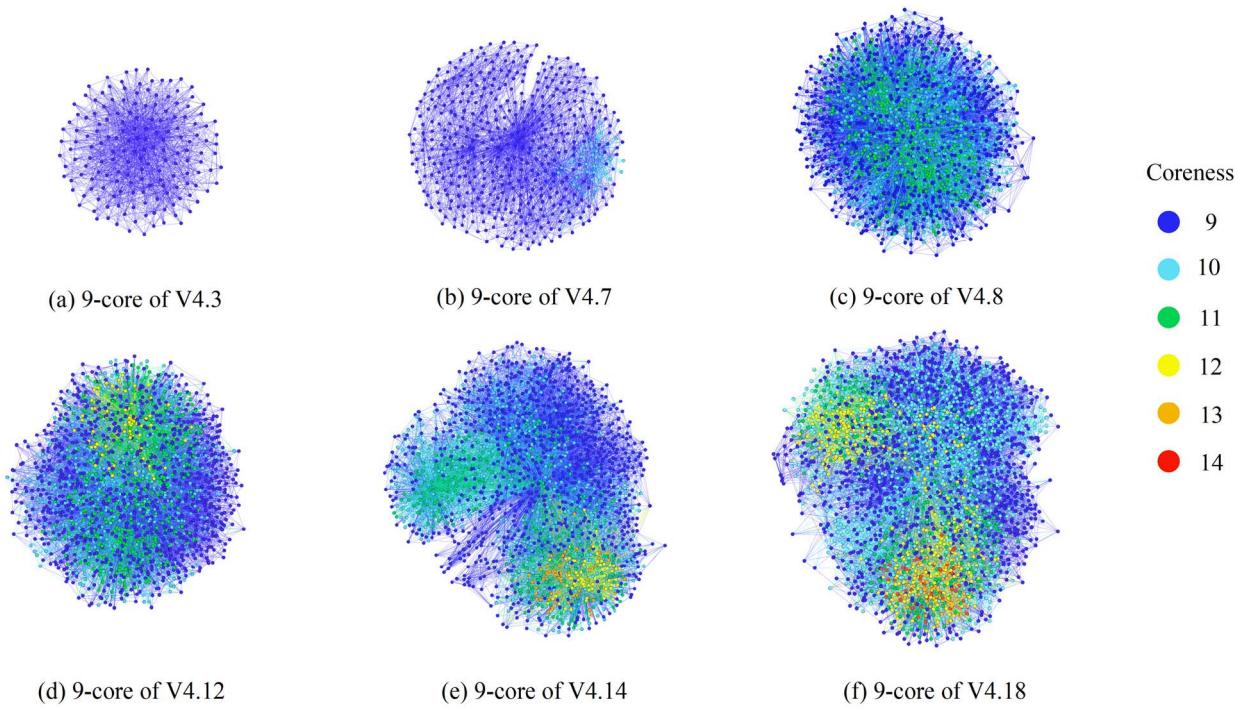


Fig. 14. Sketches of the 9-cores of kernel networks in some specific versions. These six versions containing V4.3, V.4.7, V.4.8, V.4.12, V.4.14 and V.4.18 represent some major changes during the whole evolution. The maximum of coreness is increasing from 9 to 14 and the 9-core structure becomes more complicated.

V4.8, V4.12, V4.14 and V4.18. The maximum of coreness in these kernel networks is increasing from 9 to 14. The changes of function distribution in 9-core can be obtained in Fig. 15, where the 9-cores of all versions are chosen to make module distribution analysis. In Fig. 15, the evolution of the 9-core is composed of four stages: Stage 1 (V4.0-V4.2), Stage 2 (V4.3-V4.7), Stage 3 (V4.8-V4.15) and Stage 4 (V4.16- V4.20). In general, the size of 9-core is growing with the increasing of sub-version in each stage except the rather small scale in Stage 2. The analysis of module distribution in Fig.15 suggests that this small scale in Stage 2 is mainly caused by the shrunk fs (file-system) module. It can also be obtained that the net module begins to appear in 9-core firstly in V4.8. Drivers is taking more ratio along the development process. In the next subsection, we will specify the relation between the evolution of function-call network and the real kernel structure.

C. Analysis Based on the Change-Log of Kernel

Some prominent features and update information of specific versions in Linux kernel's change-log have been provided in TABLE II in order to deeply understand the evolution of function-call network in kernel.

In version 4.3, the remove of the Ext3 file-system which is the core part of the kernel connected with many other modules actually caused the reduction of many nodes with large degree and smaller 9-core in Fig. 15. In Linux kernel, the fs module is highly connected with kernel, mm (memory management) and drivers, the core of kernel will decrease apparently once the part of *fs* module has been removed. Meanwhile, the addition of many new drivers brings a lot of functions with low degree to the kernel. The two parts confirmed before result in the sharp descending of degree and clustering coefficient in V4.3. The updated structure will naturally cause the increase of betweenness with more loads on bridge nodes, and longer shortest path length.

In version 4.8, Linux kernel supports for the transparent Huge Pages in the page cache and eXpress Data Path. These two new features mainly influence the fs, mm, kernel and net modules by adding new functions and calls. Although some central parts become larger and more complex, this growth

TABLE II
[49] PROMINENT FEATURES FOR SOME TYPICAL VERSION OF KERNEL

Version of Kernel	Prominent Features and Update Information
V4.3	Remove: the Ext3 filesystem (and the Ext4 remained for upgrading of filesystem); many new drivers
V4.8	Addition: transparent Huge Pages in the page cache; eXpress Data Path (a high performance, programmable network data path); XFS reverse mapping (the building block of several upcoming features); many new drivers
V4.16	Addition: jailhouse hypervisor (a static partitioning hypervisor that runs bare metal but cooperates closely with Linux); many new drivers

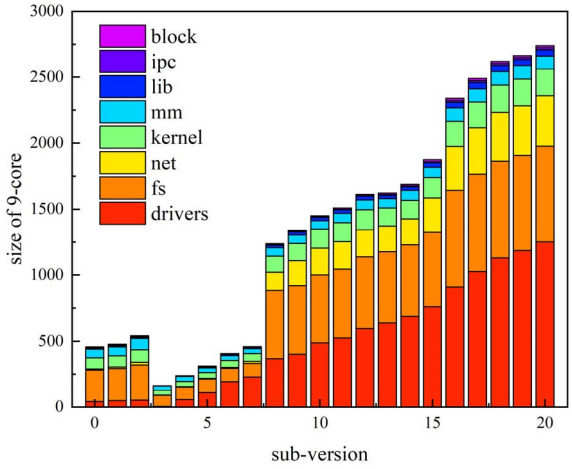


Fig. 15. The evolution of module distribution in 9-core of function-call networks. Each column represents a 9-core structure in corresponding version and is divided into eight parts with different colors to show the module distribution. The eight parts is respectively block, ipc (Inter-process communication), lib (dependent libraries), mm (memory management), kernel, net, fs (file system) and drivers.

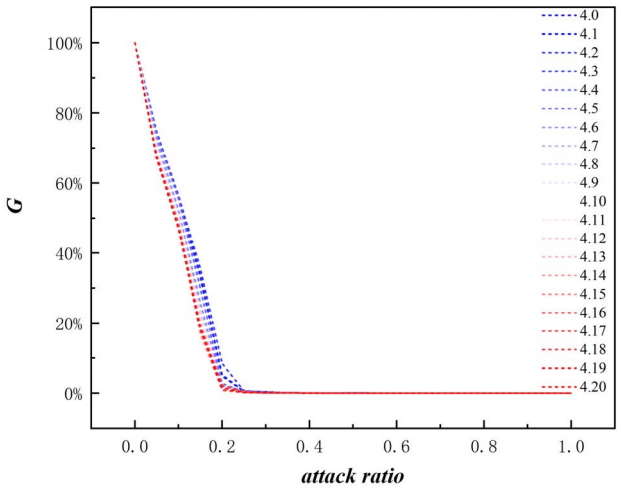


Fig. 16. The evolution of network percolation analysis in all kernel versions. Each dotted curve with different color represents a kernel network of specific version. G is the proportion of maximal connected component in the whole function-call network and attack ratio is the percentage of the nodes we remove in the network based on degree sequence. The trends of all curves seem similar.

compared to that in drivers distributed in regional part of kernel is rather low, which also indicates the heterogeneous growth of kernel network. This analysis can explain the substantial increase of average degree but slight promotion of clustering coefficient.

In version 4.16, there is an initial platform support added for the jailhouse hypervisor. Jailhouse hypervisor is a static partitioning hypervisor that runs bare metal but cooperates closely with Linux. Since the jailhouse is a hypervisor of running kernel, it needs many interactions with the central parts of different modules in kernel. It might be a part of the reason resulting in the fact that the scale of 9-core is increasing with the growth of each major module in V4.16.

The above analysis demonstrates that the model of kernel in Android OS using complex network theory can reflect the practical effect of development changes in a global view. It is expected to be utilized for the analysis of current or future software systems which are difficult to analyze by traditional methods.

D. Evolutions of Reliability Based on Percolation

In this subsection, the percolation analysis of all 21 kernel networks has been implemented. The result of degree-based attack is shown in Fig. 16. The system connectivity is defined as the proportion of the maximal connected component G. The trend of connectivity of all kernel networks is surprisingly similar. This may reflect the hidden stable pattern of Linux kernel over many versions.

VI. Conclusion

In this paper, we focus on the analysis of network reliability and evolution features in software system. To deeply understand the software network in Android OS, the percolation method is applied to analyze the function-call graph in different granularity. In this way, the results show nodes with higher strength seems more important to system reliability in community-interaction network. For the community of software network, we found three different relations between the topological properties and the community size. Moreover, we found that although topological properties of kernel network is evolving with different released versions, percolation properties seems rather stable. The result containing the evolution of topological networks, k-core decomposition and percolation analysis is highly connected with the practical applications in kernel of different versions, which can be used to support reliability testing analysis of current or future software system. This finding may also help the system design to understand the different roles of functions in the architecture, from the viewpoint of the whole software system. Our future directions mainly refer to the correlation analysis between major faults of kernel versions, system reliability models and the investigation of kernel healthy status in network perspective.

References

[1] K.-Y. Cai and B.-B. Yin "Software execution processes as an evolving complex network," *Inf. Sci.* vol. 179, no. 12, pp. 1903–1928, May 2009.

[2] W. M. Elsharkasy, A. Khajeh, A. M. Eltawil, and F. J. Kurdahi, "Reliability enhancement of low-power sequential circuits using reconfigurable pulsed latches," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 7, pp. 1803–1814, Jul. 2017.

[3] V. M. Van Santen, J. Martin-Martinez, H. Amrouch, M. M. Nafria, and J. Henkel, "Reliability in super- and near-threshold computing: A unified model of RTN, BTI, and PV," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 293–306, Jan. 2018.

[4] C. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.* vol. 68, no. 4, 2003, Art. no. 046116.

[5] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 687–708, Oct. 2007.

[6] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *Trans. Softw. Eng. Methodol.*, vol. 18, no. 1, pp. 1–26, Sep. 2008.

[7] S. Valverde, R. F. Cancho, and R. V. Solé, "Scale-free networks from optimal design," *Europhys. Lett.*, vol. 60, no. 4, pp. 512–517, Nov. 2002.

[8] G. Hardy, C. Lucet, and N. Limnios, "K-terminal network reliability measures with binary decision diagrams," *IEEE Trans. Rel.*, vol. 56, no. 3, pp. 506–515, Sep. 2007.

[9] A. Volkanovski, M. Cepin, and B. Mavko, "Application of the fault tree analysis for assessment of power system reliability," *Rel. Eng. Syst. Saf.*, vol. 94, no. 6, pp. 1116–1127, Jun. 2009.

[10] D. Li, Q. Zhang, E. Zio, S. Havlin, and R. Kang, "Network reliability analysis based on percolation theory," *Rel. Eng. Syst. Saf.*, vol. 142, pp. 556–562, Oct. 2015.

[11] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2010, pp. 447–456.

[12] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: General security support for the Linux kernel," *Found. Intrusion Tolerant Syst.*, Los Alamitos, CA, USA, 2003, pp. 213–226.

[13] W. Gu, Z. Kalbarczyk, Ravishankar, K. Iyer, and Z. Yang, "Characterization of Linux kernel behavior under errors," in *Proc. Int. Conf. Dependable Syst. Netw.*, San Francisco, CA, USA, Jun. 2004, pp. 459–468.

[14] P. A. Erdős and Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen* vol. 6, no. 1, pp. 290–297, 1959.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature* vol. 393, pp. 440–442 Jun. 1998.

[16] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[17] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the World-Wide Web," *Nature*, vol. 401, pp. 130–131, Sep. 1999.

[18] Y. Xia, M. Small, and J. Wu, "Introduction to focus issue: Complex network approaches to cyber-physical systems," *Chaos*, vol. 29, no. 9, Sep. 2019, Art. no. 093123.

[19] D. Liu and C. K. Tse, "Cascading failure of cyber-coupled power systems considering interactions between attack and defense," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4323–4336, Nov. 2019.

[20] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," *Phys. Rev. Lett.*, vol. 86, no. 14, pp. 3200–3203, Jul. 2002.

[21] S. Meloni, A. Arenas, and Y. Moreno, "Traffic-driven epidemic spreading in finite-size scale-free networks," *Proc. Nat. Acad. Sci. USA*, vol. 106, no. 40, pp. 16897–16902, Oct. 2009.

[22] G. Zeng *et al.*, "Switch between critical percolation modes in city traffic dynamics," *Proc Nat. Acad. Sci. USA*, vol. 116, no. 1, pp. 23–28, Jan. 2019.

[23] L. Zhang, G. Zeng, D. Li, H.-J. Huang, H. E. Stanley, and S. Havlin, "Scale-free resilience of real traffic jams," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 18, pp. 8673–8678, Apr. 2019.

[24] D. Li *et al.*, "Percolation transition in dynamical traffic network with evolving critical bottlenecks," *Proc. Nat. Acad. Sci. USA*, vol. 112, no. 3, pp. 669–672, Jan. 2015.

[25] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, "Controllability of complex networks," *Nature*, vol. 473, no. 7346, pp. 167–173, May 2011.

[26] G. Yan, G. Tsekenis, B. Barzel, J.-J. Slotine, Y.-Y. Liu, and A.-L. Barabási, "Spectrum of controlling and observing complex networks," *Nature Phys.*, vol. 11, no. 9, pp. 779–786, Sep. 2015.

[27] Y. Song, D. J. Hill, and T. Liu, "On extension of effective resistance with application to graph laplacian definiteness and power network stability," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp. 4415–4428, Nov. 2019.

[28] Z. Chen, J. Wu, Y. Xia, and X. Zhang, "Robustness of interdependent power grids and communication networks: A complex network perspective," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 1, pp. 115–119, Jan. 2018.

[29] J. Wu, J. Zhong, Z. Chen, and B. Chen, "Optimal coupling patterns in interconnected communication networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 8, pp. 1109–1113, Aug. 2018.

[30] X. Wu, W. Wei, L. Tang, J. Lu, and J. Lu, "Coreness and h-index for weighted networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 8, pp. 3113–3122, Aug. 2019.

[31] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proc. Nat. Acad. Sci. USA*, vol. 101, no. 11, pp. 3747–3752, Mar. 2004.

[32] L. Wen, D. Kirk, and R. G. Dromey, "Software systems as complex networks," in *Proc. 6th IEEE Int. Conf. Cogn. Inform. (ICCI)*, 2007, pp. 106–115.

[33] S. Jenkins and S. Kirk, "Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution," *Inf. Sci.*, vol. 177, no. 12, pp. 2587–2601, Jun. 2007.

[34] W. Pan, B. Li, Y. Ma, and J. Liu, "Multi-granularity evolution analysis of software using complex network theory," *J. Syst. Sci. Complex*, vol. 24, no. 6, pp. 1068–1082, Dec. 2011.

[35] L. Šubelj and M. Bajec, "Community structure of complex software systems: Analysis and applications," *Phys. A, Stat. Mech. Appl.*, vol. 390, no. 16, pp. 2968–2975, Aug. 2011.

[36] M. A. Fortuna, J. A. Bonachela, and S. A. Levin, "Evolution of a modular software network," *Proc. Nat. Acad. Sci. USA*, vol. 108, no. 50, pp. 19985–19989, Dec. 2011.

[37] X. Zheng, "Analysis on key nodes behavior for complex software network," in *Proc. Int. Conf. Inf. Comput. Appl. (ICICA)*, 2012, pp. 59–66.

[38] K.-K. Yan, G. Fang, N. Bhardwaj, R. P. Alexander, and M. Gerstein, "Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 20, pp. 9186–9191, May 2010.

[39] X. Zheng, D. Zeng, H. Li, and F. Wang, "Analyzing open-source software systems as complex networks," *Phys. A, Stat. Mech. Appl.*, vol. 387, no. 24, pp. 6190–6200, Oct. 2008.

[40] Y. Gao, Z. Zheng, and F. Qin, "Analysis of Linux kernel as a complex network," *Chaos, Solitons Fractals*, vol. 69, pp. 246–252, Dec. 2014.

[41] H. Wang, Z. Chen, G. Xiao, and Z. Zheng, "Network of networks in Linux operating system," *Phys. A, Stat. Mech. Appl.*, vol. 447, pp. 520–526, Apr. 2016.

[42] G. Xiao, Z. Zheng, and H. Wang, "Evolution of Linux operating system network," *Phys. A, Stat. Mech. Appl.*, vol. 466, pp. 249–258, Jan. 2017.

[43] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Netw.*, vol. 1, no. 3, pp. 215–239, Jan. 1978.

[44] G. Fagiolo, "Clustering in complex directed networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.* vol. 76, no. 2, Aug. 2007, Art. no. 026107.

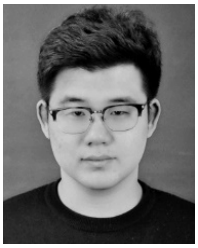
[45] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "K-core decomposition: A tool for the visualization of large scale networks," 2005, *arXiv:cs/0504107*. [Online]. Available: https://arxiv.org/abs/cs/0504107

[46] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, vol. 2008, no. 10, Oct. 2008, Art. no. P10008.

[47] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. E69, Feb. 2004, Art. no. 026113.

[48] A. Barrat, M. Barthélemy, and A. Vespignani, "Modeling the evolution of weighted networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 70, no. 6, 2004, Art. no. 066149.

[49] *LinuxVersions*. [Online]. Available: https://kernelnewbies.org/LinuxVersions



Anzhuo Yao received the B.E. degree from Harbin Engineering University, Harbin, China, in 2018. He is currently pursuing the master's degree with the School of Reliability and Systems Engineering, Beihang University, Beijing, China. He is currently involved in the field of reliability in software systems and complex network theory.

Pengfei Sun received the B.E. degree in automation from the College of Information Science and Engineering, Northeastern University, Shenyang, China, in 2017. He is currently pursuing the master's degree in control science and engineering with the School of Reliability and Systems Engineering, Beihang University, Beijing, China. His research interests include reliability analysis, and complex network theory and its applications.

Shunkun Yang (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in reliability and systems engineering from Beihang University, Beijing, China, in 2000, 2003, and 2011, respectively.

From 2016 to 2019, he was an Associate Research Professor with the School of Reliability and Systems Engineering, Beihang University. His research interests include complex network-based software testing and reliability analysis.

Daqing Li (Member, IEEE) received the Ph.D. degree from Bar-Ilan University, Israel, in 2011.

He is currently a Full Professor at Beihang University, China, where he initiated the laboratory of complex system reliability. Focusing on the reliability engineering of complex system, he has made novel contributions in the modeling of system failure, identification of risk propagation, and corresponding reliability management.