Chinese Journal of Aeronautics, (2019), 32(1): 176–185

Chinese Society of Aeronautics and Astronautics
& Beihang University

Chinese Journal of Aeronautics

cja@buaa.edu.cn
www.sciencedirect.com

Evolution analysis of a UAV real-time operating system from a network perspective

Zheng ZHENG, Guanping XIAO *

School of Automation Science and Electrical Engineering, Beihang University, Beijing 100083, China

Received 24 November 2017; revised 28 February 2018; accepted 12 April 2018
Available online 26 April 2018

KEYWORDS

Complex networks;
Evolution;
FreeRTOS;
k-core decomposition;
Real-time operating system

Abstract With the flourishing development of Unmanned Aerial Vehicles (UAVs), the mission tasks of UAVs have become more and more complex. Consequently, a Real-Time Operating System (RTOS) that provides operating environments for various mission services on these UAVs has become crucial, which leads to the necessity of having a deep understanding of an RTOS. In this paper, an empirical study is conducted on FreeRTOS, a commonly used RTOS for UAVs, from a complex network perspective. A total of 85 releases of FreeRTOS, from V2.4.2 to V10.0.0, are modeled as directed networks, in which the nodes represent functions and the edges denote function calls. It is found that the size of the FreeRTOS network has grown almost linearly with the evolution of the versions, while its main core has evolved steadily. In addition, a *k*-core analysis-based metric is proposed to identify major functionality changes of FreeRTOS during its evolution. The result shows that the identified versions are consistent with the version change logs. Finally, it is found that the clustering coefficient of the Linux OS scheduler is larger than that of the FreeRTOS scheduler. In conclusion, the empirical results provide useful guidance for developers and users of UAV RTOSs.

1. Introduction

Recently, research and applications of Unmanned Aerial Vehicles (UAVs) have become issues of intense interest. Meanwhile, missions conducted by UAVs are more and more complex, which leads to complex on-board application designs. Therefore, using a Real-Time Operating System (RTOS) to develop these complex on-board applications would be preferable and critical. An RTOS is a type of Operating System (OS) that is designed to provide real-time applications with several basic supports, such as scheduling, synchronization, resource management, precise timing, communication, and I/O.[1] Its reliability would have direct impacts on safety operations of UAVs. Therefore, analyzing a commonly used RTOS of UAVs is essential and useful for developing UAV on-board software.

Ranging from nature to human society, many complex systems can be abstracted as networks, in which nodes represent fundamental elements and edges denote interactions between fundamental elements.[2] Therefore, graph theory and network analysis methodologies can be implemented to investigate the

* Corresponding author.
E-mail address: gpxiao@buaa.edu.cn (G. XIAO).

Peer review under responsibility of Editorial Committee of CJA.

ELSEVIER Production and hosting by Elsevier

https://doi.org/10.1016/j.cja.2018.04.011

topological characteristics of complex systems. Firstly, regular lattices are utilized to analyze the network topologies of real systems. Nevertheless, its application domain is too limited. In 1959, Erdös and Rényi proposed the random network model, known as the ER network, which was constructed by randomly connecting nodes.[3] This model has dominated network research for decades. However, since the ER network is static, it could not explain some phenomena (e.g., Matthew effect[4]) that exist in ubiquitous dynamic evolution systems. Due to the introductions of the small-world network model and the scale-free network model at the end of the past century,[5,6] network science has greatly developed and attracted various research studies on real-world complex systems, such as protein networks,[7] Internet,[8] and transportation networks.[9,10] At the same time, several research directions have been flourishing, including traffic dynamics,[11] multi-layer networks,[12,13] optimization processes,[14] network control,[15,16] and so on. One interesting aspect is to analyze software systems from a complex network perspective.[17]

Large-scale software systems represent one of the most complex artificial systems. By using complex network theory, a wealth of studies on the analysis of software structures and functions has been conducted. Ref.[18] shows, for the first time, that software networks also present scale-free and small-world behaviors, which is in accordance with other real-world complex systems. In Ref.[19], researchers studied the network characteristics of two different representations of interactions among software components: call graphs and class diagrams. In addition, they proposed a refactoring process-based software evolution model. Moreover, several static analyzers have been developed to translate programs into graph representation for software understanding.[20,21] Among various types of software systems, operating systems (OSs) are typical and critical, and thus have attracted a large amount of attention from a network perspective. Zheng et al.[22] proposed two network growth models for describing the development of Gentoo Linux. Gao et al.[23] investigated the kernel directory of the Linux kernel as a complex network. It was found that when encountering intentional attacks, having large in-degree nodes would lead to more damage to the whole system. Wang et al.[24] explored the coupling relationships among components in Linux, and analyzed the impact of system failures on networks. Recently, Xiao et al.[25] studied the evolution of 62 major releases of the Linux kernel, ranging from versions 1.0 to 4.1, from a network perspective. This work revealed the characteristics of the structure and functionality evolution of the Linux network. However, most of the research has focused on general-purpose operating systems (GPOSs, e.g., Linux). Studies that analyze real-time operating systems (RTOSs) from a network perspective are still rarely performed.

The greatest difference between an RTOS and a GPOS is that an RTOS has a deterministic scheduler, which means that it must provide a timely response to real-world events. Among various types of RTOSs, FreeRTOS is an open-source, market-leading, and UAV commonly used RTOS.[26,27] For example, the firmware of Crazyflie 2.0, a popular open-source flying development platform, is based on it.[28] FreeRTOS officially supports approximately 35 architectures that cover more than 20 vendors and a hundred types of processors. Recent research on FreeRTOS has concentrated on functionality extension,[29] testing and verification,[30,31] scheduler analysis,[32] and so on. As a commonly used RTOS for embedded systems, it is very interesting to explore the evolution of FreeRTOS and its relationship with a real-time system's properties, such as small storage space support, various architectural designs, and high real-time requirement.

Our study is performed with 85 releases of FreeRTOS, from V2.4.2 to V10.0.0. In this study, we focus on the following four research questions, which can reflect the topological and functional structure evolution of FreeRTOS comprehensively.

RQ1. How have topological properties of the FreeRTOS network evolved over versions?

According to Lehman's laws of software evolution,[33] the evolution of software could follow several laws, such as continuing growth, increasing complexity, and conservation of organizational stability. Therefore, from a complex network perspective, how FreeRTOS has evolved over versions and what the manifestations of the topological properties during the evolution are, as well as a comparison with the evolution of the Linux OS network, are interesting subjects to explore.

RQ2. How has the k-core structure of the FreeRTOS network evolved over versions?

Identifying highly interconnected subgraphs of a graph as well as finding characteristics of these substructures attracts topical interest in network research.[2] Among various metrics that are used to evaluate the relative importance of nodes, using k-cores is a well-established method.[34] A k-core of a network can be obtained by recursively removing nodes with a degree lower than k, until all nodes in the remaining network have a degree higher than or equal to k.[35] By adopting the k-core decomposition method on FreeRTOS networks, how the k-core structures have evolved over versions will be answered in this research.

RQ3. How can we evaluate the functionality changes of FreeRTOS?

Over the past ten years, dozens of FreeRTOS versions have been released, and several functionality changes have been implemented in the operating system (e.g., new feature introductions and enhancements). Thus, network metrics that can be utilized to evaluate the functionality changes of FreeRTOS will be demonstrated.

RQ4. How has the real-time scheduler of FreeRTOS evolved over versions?

As the most fundamental composition of an RTOS, how the real-time scheduler of FreeRTOS has evolved with versions and the nature of the discrepancy between a real-time scheduler and a non-real-time scheduler on networks will be examined in this research.

The remainder of this paper is organized as follows. Section 2 describes methods utilized in this paper. Section 3 presents analytical results and discussion. Finally, conclusions are given in Section 4.

2. Methods

2.1. Research data

FreeRTOS was initially developed by Richard Barry in 2003. Due to the open-source characteristic, all the codes of

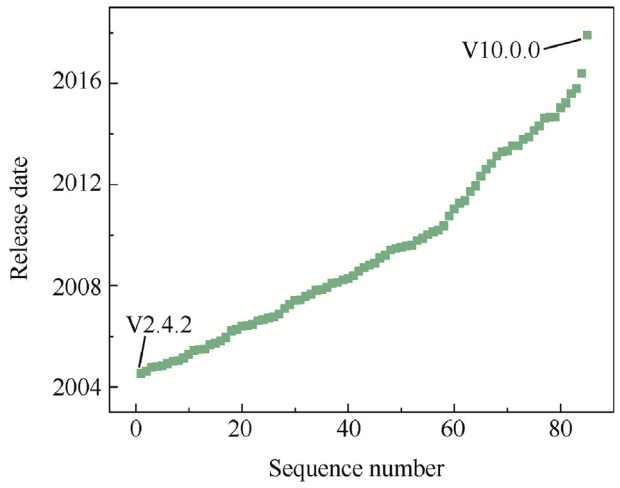


Fig. 1 A total of 85 FreeRTOS releases, ranging from V2.4.2 developed in 2004 to V10.0.0 in 2017.

FreeRTOS for the 85 releases, from V2.4.2 to V10.0.0, are available on the SourceForge.[A] Since each version of FreeRTOS was successively developed based on its previous version, we chose the 85 releases as the research data for the evolution study. The selected versions and their release dates are shown in Fig. 1. Note that SN represents the version's sequence number, according to its release date (e.g., the SN of V2.4.2 is 1 and that for V10.0.0 is 85). This symbol will be used and discussed in the following parts of the paper. In addition, the mapping between major version numbers (e.g., V3.0.0) and their sequence numbers is exhibited in Table 1.

Compared with most GPOSs, the size of the FreeRTOS source code is small. For example, the lines of code (LOCs) of the FreeRTOS release V10.0.0 are approximately 70000, while the LOCs of the Linux kernel V4.1 are more than 19 million. Fig. 2 displays the structure of the FreeRTOS V10.0.0 source code. For all FreeRTOS versions, the core code is contained in three files, which are called *tasks.c, list.c*, and *queue.c*. Scheduler functionalities are mainly implemented in the file *tasks.c*. Structures and functions used by the scheduler are defined in the file *list.c*, while the file *queue.c* contains thread-safe queues that are implemented for synchronization and inter-task communication. In addition to the core code files, there are four optional files, called *croutine.c, timers.c, event_groups.c*, and *stream_buffer.c*, which implement co-routine functionality, software timer, event group, and stream buffers, respectively. These four files are added in V4.0.0, V7.0.0, V8.0.0, and V10.0.0, separately. The directory *include* contains the core header files. The directory *portable* includes the architecture-dependent code. In this directory, each compiler contains an architecture-specific code called *port.c* for the supported processor architecture. Since memory management is specifically defined for different architectures, FreeRTOS keeps the memory management API in the portable layer, and implements several heap management samples in the directory *MemMang* that can be used for most of the architectures.[36]

[A] https://sourceforge.net/projects/freertos. The latest release is V10.0.0, when we conducted this study.

2.2. Software network modeling

FreeRTOS is implemented mostly based on the C programming language, with a few functions written in an assembler language, which is used for architecture-specific details. For a C language-developed software system, its realization mainly depends on function calls, which can be commonly regarded as a call graph, as depicted in Fig. 3. We define the call graph as a directed network $G(N, E)$, where $N = \{v_1, v_2, \ldots v_n\}$ is the set of n nodes, which are in terms of functions, and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of m edges, each of which, $e_i = (v_s, v_t)$ $(i = 1, 2, \ldots, m)$, represents the call between each pair of nodes v_s and v_t $(v_s, v_t \in N)$. We model all of the 85 releases from V2.4.2 to V10.0.0 as directed networks and analyze the largest weakly connected component of each network. The definitions of the other network properties used in this study are as follows.

Clustering coefficient C: the clustering coefficient C describes the probability that a node's neighbors are also the neighbors of one another. A larger clustering coefficient means that there exist more tightly connected neighborhoods. Given a directed network, the clustering coefficient of node i is defined as[37]

$$C_i = \frac{1}{2}\frac{\sum_j\sum_k (a_{ij} + a_{ji})(a_{ik} + a_{ki})(a_{jk} + a_{kj})}{(k_i^{\text{in}} + k_i^{\text{out}})(k_i^{\text{in}} + k_i^{\text{out}} - 1) - 2\sum_j a_{ij}a_{ji}} \tag{1}$$

where $a_{ij} = 1$ if there is an edge from node i to j; otherwise, $a_{ij} = 0$. Here, k_i^{in} and k_i^{out} are the in-degree and out-degree of node i, respectively. For the whole network, the clustering coefficient is denoted as

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{2}$$

Degree distribution $P(k)$: the degree distribution is utilized to denote the probability of a randomly selected node with a degree of k.

2.3. k-core decomposition

Considering a graph $G(N, E)$ of $|N| = n$ nodes and $|E| = m$ edges, the definitions of k-core metrics are as follows.[38] k-core: a subgraph $H = (V, E|V)$ induced by the set $V \subseteq N$ is a k-core or a core of order k if and only if $\forall v \in V : k_v \geqslant k$, and H is the maximum subgraph with this property.

Coreness: node i has a coreness of k if it is located in the k-core but not in the $(k + 1)$ -core. The maximum coreness $k_{\max}$ is that the $k_{\max}$ -core is not empty, but the $(k_{\max} + 1)$ -core is. Note that $k_{\max}$ is also regarded as the graph coreness, while the $k_{\max}$ -core is usually called the main core of a graph.

Core size: it is the number of nodes of the k-core.k-shell: a k-shell S_k has all the nodes whose coreness is k. The k-core is thus composed of all S_c with $c \geqslant k$.

Fig. 4 shows a sketch of the k-core decomposition for an example network. Different types of closed lines separate different cores, while different colors of nodes represent different corenesses. All of the nodes of this network belong to the 1-core. After recursively removing all of the nodes with a degree less than 2, the remaining nodes compose the 2-core. Finally, there is the 3-core, which is the innermost set of nodes. It

Table 1 Mapping between major version numbers and their sequence numbers.

Major version	3	4	5	6	7	8	9	10
Sequence number	10	18	40	53	61	75	84	85

Note: for example 3 means V3.0.0 major version.

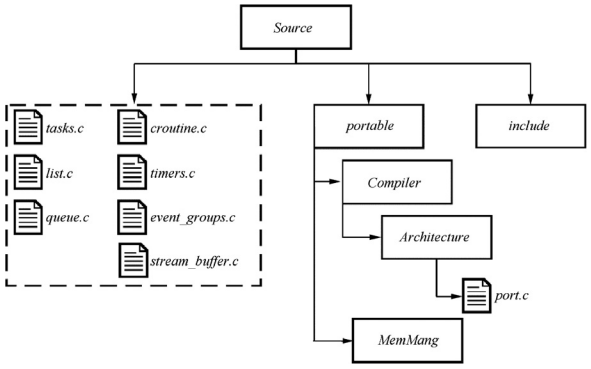


Fig. 2 Structure of FreeRTOS V10.0.0 source code.

should be noted that the degree of a node could not represent the hierarchy. For example, as shown in Fig. 4, the degree of node a is 4, while its coreness is 1. Comparatively, the degree of node b is 3, but its coreness is 3.

3. Results and discussion

In this section, we present the evolution analysis results of the FreeRTOS network from four aspects. These aspects are the network property evolution, the k-core structure evolution, the functionality change evaluation, and the real-time scheduler evolution. The four parts correspond to the four research questions illustrated in Section 1.

3.1. Evolutions of network properties

In this subsection, we present the analysis of the topological property evolution of the FreeRTOS network. According to Lehman's laws of software evolution, in continuing growth,[33] the functionality of a program must be continually increased to satisfy user feature requirements during its lifecycle. The continuing growth could be validated by calculating software size metrics and determining their trends over time. Obviously, as exhibited in Fig. 5(a) and (b), the size of the FreeRTOS network grows almost linearly with increasing sequence numbers. It can be observed that the numbers of nodes n and edges m in V10.0.0 are approximately 7.36 and 8.00 times higher than those of V2.4.2, respectively. This finding indicates that the functionality of FreeRTOS is continually growing and is in accordance with law 6 of Lehman's laws of software evolution.

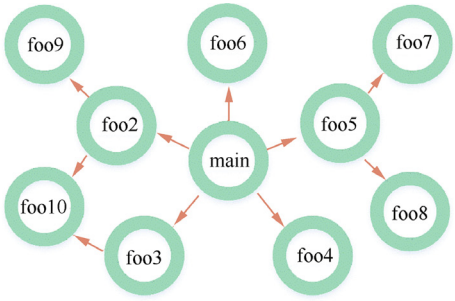
Fig. 5(c) depicts that the clustering coefficient C evolves with an increase in the sequence numbers. It can be easily obtained that the evolution of the clustering coefficient C has two stages, i.e., decreasing from V2.4.2 to V6.1.1 and then increasing from V6.1.1 to V10.0.0. The decreasing of C means that the local connections of the FreeRTOS network would tend to be looser with evolution. In contrast, the increasing of C in the second stage indicates that the local connections become tighter. This phenomenon might be attributed to the reason that developers tend to pay more attentions to improve the core functionality of FreeRTOS other than to port more supported architectures. By counting the supported compiler directories in the portable layer of the source code, it is found that 10 new compilers have been added from V2.4.2 to V6.1.1, while only 2 new compilers from V6.1.1 to V10.0.0.

The evolution of the average degree $\langle k \rangle$ (i.e., average out-/in-degree) is shown in Fig. 5(d). Similar to the clustering coefficient C, a two-stage evolution trend is found. Fig. 5(e) and (f) presents the degree distributions of V2.4.2 and V10.0.0. It can be notably observed that the degree distributions of the two versions are similar: the out-degree distribution experiences an exponential distribution, while the in-degree distribution follows a power-law distribution. This finding is in accordance with the results on Linux OS.[23–25] Both the out-degree and in-degree are quite heterogeneous, which indicates that in the software development process, only a few functions would have many calls or would be called many times, due to the requirements of reliability and maintenance.

Moreover, we make a comparison between the network evolutions of FreeRTOS and Linux OS, as shown in Table 2. Note that the data on Linux in Ref. [25] has 62 major releases, including versions 1.0 to 4.1. It can be observed that the sizes of the two networks are significantly different, which is attributed to their different application domains. More functionalities, e.g., device drivers and filesystems, are obviously required by GPOSs (e.g., Linux OS) users. However, RTOSs are typically

```
#include<stdio.h>
//a_C_language_program.c
void foo2() {foo9(); fool0();}
void foo3() {foo10();}
void foo5() {foo7(); foo8();}
int main() {foo2(); foo3();
            foo4(); foo5();
            foo6(); return 0;
}
```

(a) Source code of a C language program



(b) Call graph

Fig. 3 A depiction for modeling a C language program as a directed network.

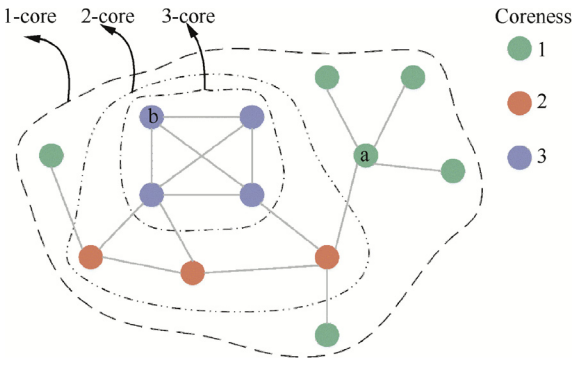


Fig. 4 Illustration of k-core decomposition for an example network.

used in embedded systems, which usually have a small storage space. This circumstance would limit the sizes of RTOSs. In addition, the functionality of RTOSs is more focused on task scheduling. Moreover, the variations in the average degrees (i.e., average out-/in-degree) and clustering coefficients of both systems are of the same order of magnitude. In addition, the degree distributions of the out-degree and in-degree for the two systems follow the same type of distributions.

3.2. Evolution of k-core structures

In this part, we analyze the evolution of k-core structures of the FreeRTOS network using the k-core decomposition method introduced in Section 2.3. Fig. 6 shows the k-core and its decomposition process of the FreeRTOS V10.0.0 network. Colors on the nodes distinguish different k-shells and the process of k-core decomposition of V10.0.0 from 1 to 6. Note that all the figures are plotted by cytoscape, a network visualization tool. Notably, it can be observed that with increasing k, the inner part of the network remains connected and becomes more internal, which indicates the hierarchy structure of the FreeRTOS network. Furthermore, as depicted in Fig. 6, the core size decreases gradually with increasing k, which reflects the centrality of the structure. For studying the correlation between the core size and coreness, we plot the core sizes with increasing corenesses in Fig. 7. This figure shows a power-law relation between the core size and coreness, which illustrates that each k-core structure consists of a constant fraction of the $(k-1)$ -core and is well in accordance with other software systems, such as MySQL and VTK.[39]

In addition, k-core decomposition can distinguish the most central parts, i.e., the maximal coreness of the network. The maximal coreness region of the network consists of the most important nodes to a certain extent, since it composes the skeleton of the software network. In the following, we concentrate on analyzing the evolution of this part, i.e., the maximal coreness of the FreeRTOS network.

Fig. 8 displays the evolution of the k-core structures of the FreeRTOS network. The maximal coreness, also called the graph coreness, grows steadily with increasing sequence numbers, as shown in Fig. 8(a). The range of the graph coreness is from 4 to 7, which is close to those of other software systems. For example, the graph coreness of Linux is 6, and that of VTK is 5.[39] Compared with other real-world complex systems, it is found that the graph coreness of the FreeRTOS network is much smaller. For example, the graph coreness of the Internet AS level is approximately 20.[40] This finding demonstrates that a reliable and readable programming method would usually not prefer to implement a much higher density and concentration of function interactions, due to the requirement of complexity controlling, which would be beneficial to development and maintenance.

The evolution of the graph coreness consists of five plateau stages: Stage A (V2.4.2-V4.6.1), Stage B (V4.7.0-V6.1.1), Stage

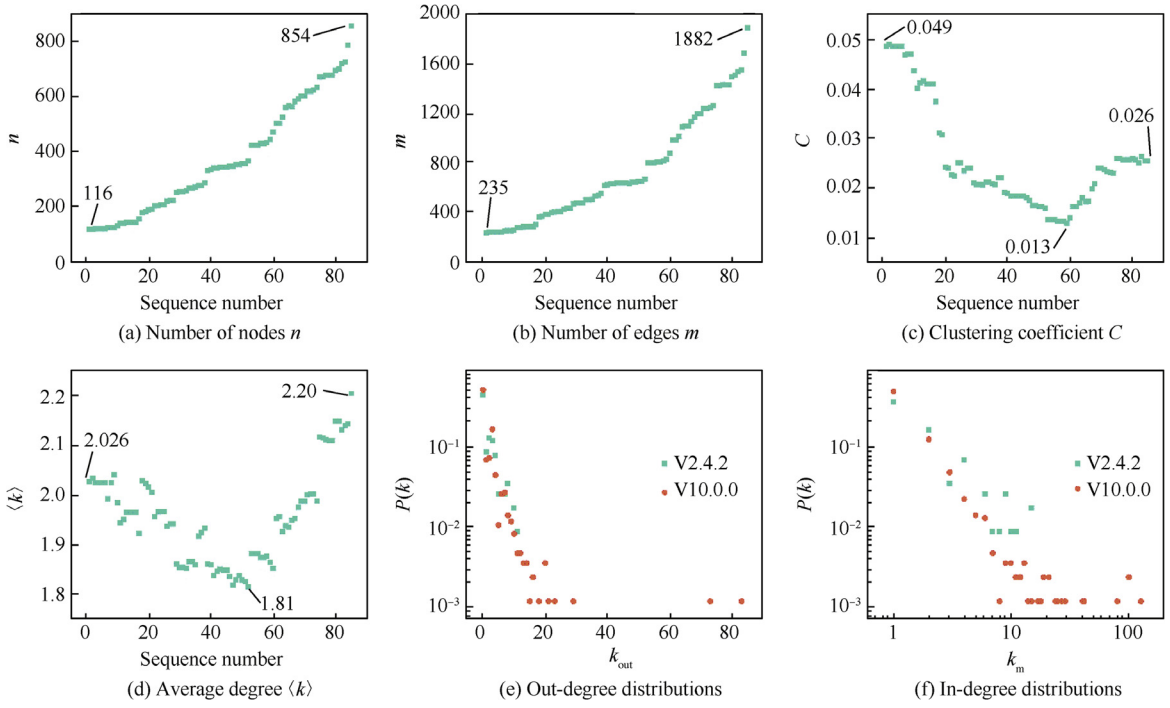


Fig. 5 Evolutions of topological properties of FreeRTOS network.

Table 2 Comparison between network evolutions of FreeRTOS and Linux OS.

OS	n	m	$\langle k \rangle$	C	Out-degree	In-degree
FreeRTOS	116–854	235–1882	1.81–2.20	0.013–0.049	Exponential	Power-law
Linux OS	3377–398092	11665–1529505	3.45–3.84	0.040–0.067	Exponential	Power-law

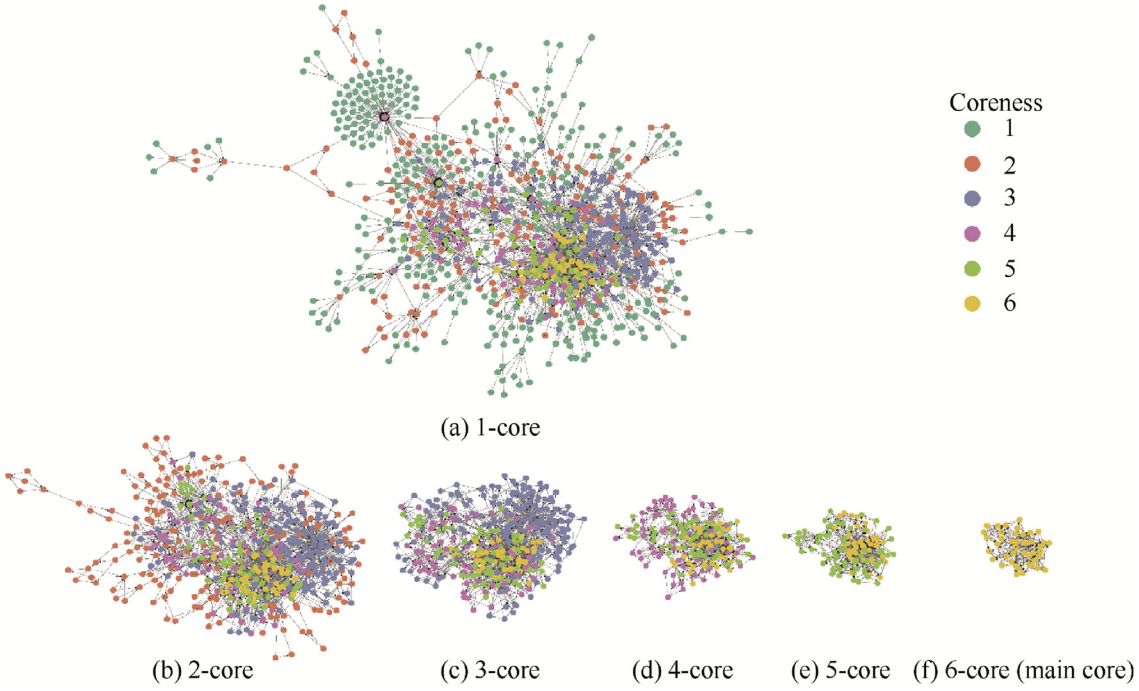


Fig. 6 Sketches of the k-cores of FreeRTOS V10.0.0.

C (V7.0.0-V8.1.2), Stage D (V8.2.0-V8.2.3), and Stage E (V9.0.0-V10.0.0). With increasing sequence numbers, the graph coreness grows from 4 in V2.4.2 to 7 in V8.2.0, which indicates that the complexity of FreeRTOS has increased. However, the graph coreness decreases to 6 in V9.0.0. This trend might be attributed to the requirement for controlling the complexity of FreeRTOS, and it is in accordance with Lehman's laws of software evolution: conservation of organizational stability.[33] Another observation that is worthwhile to mention is that by inspecting the change logs,[B] it is found that the versions of the turning points, marked by dotted lines, have major functionality changes. For example, a new software timer was implemented in V7.0.0.

Fig. 8(b) shows the size evolution of the main core with increasing sequence numbers. An interesting phenomenon is that the main core's size does not grow with an increase in the network size or the graph coreness. The core sizes of most versions are approximately 30. It can be observed that the maximal core size appears in some versions with a graph coreness of 4, while the minimal core size occurs in some versions with a graph coreness of 6. Compared with the graph coreness evolution, it is found that when the graph coreness significantly changes in some versions, the corresponding core sizes of these versions also change. This finding occurs mainly due to the activity that major functionality updates were implemented in these versions. Moreover, it can be seen that, although the graph corenesses of some versions are identical, the core sizes of these versions are significantly different. The main core is the most central and fundamental part of a network. Thus, changes in its major functionality would be inevitably reflected by changes in the network structure.

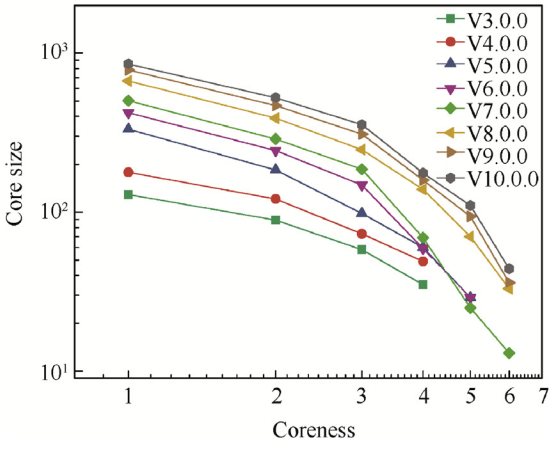


Fig. 7 Correlations between coreness and core size across different versions.

[B] http://www.freertos.org/History.txt.II

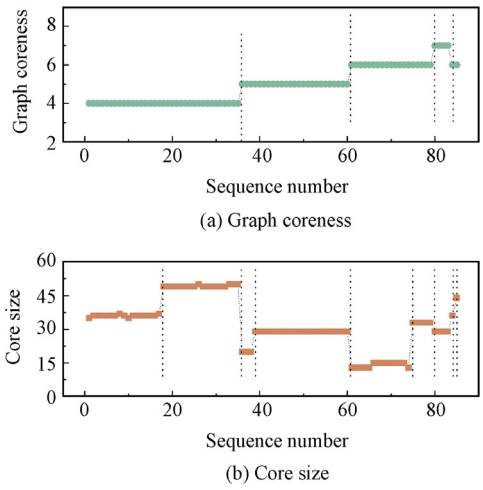


Fig. 8 Evolution of k-core structures.

3.3. Evaluation of functionality changes

During the evolutions of software systems, new feature introductions and enhancements are continuously implemented. In this subsection, to evaluate the functionality changes of FreeRTOS, we propose a metric based on k-core decomposition analysis.

From the analysis in Section 3.2, we have the following observations: (A) the evolutions of the graph coreness and core size of the FreeRTOS network could be utilized to evaluate the functionality changes that occurred during the evolution period; (B) although different versions could have an identical graph coreness, their core sizes were different. Based on the observations, we investigate the node change of the main core in detail to evaluate the functionality changes of FreeRTOS. It is noted that a node change refers to the difference of nodes in main cores between two adjacent versions. We define the following metrics for measuring the changes.

$\text{Node}^i_{\text{app}}$**:** represents the number of newly appeared nodes in the main core of version i compared with version $i-1$. For example, $\text{Node}^3_{\text{app}} = 5$ means that 5 new nodes occur in the main core of version 3 compared with version 2.$\text{Node}^i_{\text{disapp}}$**:** denotes the number of disappeared nodes in the main core of version i compared with version $i-1$. For example, $\text{Node}^3_{\text{disapp}} = 2$ means that 2 nodes disappear in the main core of version 3, but they exist in version 2.$\text{Node}^i_{\text{changed}}$**:** represents the number of changed nodes in the main core of version i compared with version $i-1$. It equals to the sum of the numbers of newly appeared and disappeared nodes, i.e.,

$$\text{Node}^i_{\text{changed}} = \text{Node}^i_{\text{app}} + \text{Node}^i_{\text{disapp}} \tag{3}$$

For example, $\text{Node}^3_{\text{changed}} = 7$ indicates that 7 nodes have changed in the main core of version 3 from version 2. Note that when $i = 1$, $\text{Node}^1_{\text{app}}$ and $\text{Node}^1_{\text{disapp}}$ are set to 0. Thus, $\text{Node}^1_{\text{changed}} = 0$.

To study the evolution of the metric, we implement a tool that is utilized to automatically inspect the main cores of the 85 versions. Fig. 9 shows the evolution of $\text{Node}_{\text{changed}}$ with increasing sequence numbers. Note that $\text{Node}_{\text{changed}} = 0$ means that the main cores of two adjacent versions are identical. It can be notably observed that $\text{Node}_{\text{changed}}$ for a few versions are significantly larger than those for the others. In this study, the evaluation threshold τ is set to 5. By analyzing the change logs of the 85 versions, it is identified that versions with $\text{Node}_{\text{changed}}$ larger than or equal to the threshold have implemented more major functionality changes, including new features, API implementations, kernel changes, and so on. A list of the identified versions is exhibited in Table 3. For example, mutex functionality was added in V4.5.0, and task notifications were introduced in V8.2.0.

Note that when evaluating main core changes, we use words "appeared" and "disappeared", not "added" and "removed". The reason is that the newly appeared nodes in the main core of version i could have already existed in version $i-1$. The emergence of these nodes might be attributed to the function calling connection changed in the main core or other shells.[41] Similarly, the nodes that disappeared in version i could not indicate that these nodes have been removed from its previous version.

It is also worthwhile to mention that versions with $\text{Node}_{\text{changed}} = 0$ could not indicate that the functions of these versions have not been changed. Through inspecting the change logs, it is found that changes that occur in versions with $\text{Node}_{\text{changed}} = 0$ are mostly architecture port supports and improvements as well as fixes. In addition, it should be noted that the threshold setting could influence the identification of versions with major functionality changes.

Furthermore, by comparing the results between Fig. 8 and Fig. 9, it can be easily observed that the proposed metric $\text{Node}_{\text{changed}}$ is more effective for evaluating the major functionality changes of FreeRTOS than using the graph coreness and core size.

3.4. Evolution of real-time scheduler

In the following, we firstly analyze the function evolution of the real-time scheduler for FreeRTOS. In addition, we build the scheduler networks of FreeRTOS and Linux OS, with the objective of comparing the discrepancy between the real-time and non-real-time schedulers from complex network points of view.

In FreeRTOS, the execution thread is called a 'task'. The functionality of the real-time scheduler, also called task management, is mainly implemented in the file *tasks.c*. Fig. 10 exhibits the evolution of the task function nodes. From the development manual,[36] it is known that the implementation of FreeRTOS has its own coding style. For example, function names are prefixed with both the file they are defined within and the type they return (e.g., vTaskSuspend() is defined in the file *tasks.c* and returns a void.). We conduct a statistic of the defined functions that correspond to the task management in the file tasks.c and plot an evolution curve. As shown in Fig. 10(a), the number of function nodes grows with increasing sequence numbers, which indicates that the functionality of the real-time scheduler is continually enhanced with the evolution of versions. In addition, although the number of task function nodes increases, it can be observed from Fig. 10(b) that the proportion of task function nodes in the entire network has

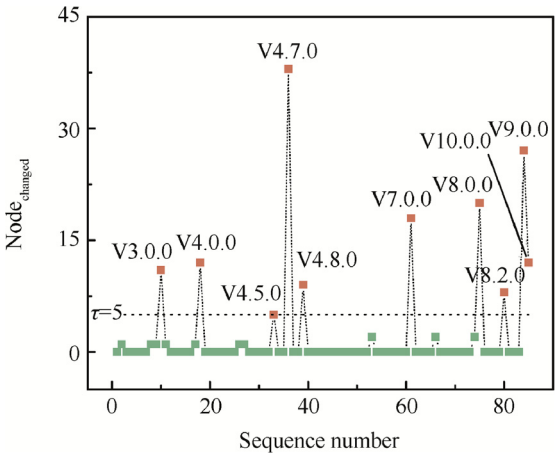


Fig. 9 Evolution of $Node_{changed}$ among 85 versions.

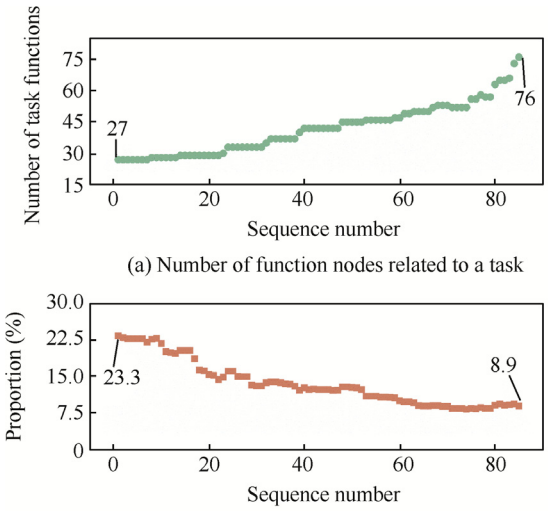


Fig. 10 Evolution of task function nodes.

Table 3 The identified versions and their major functionality changes.

Identified versions	Major functionality changes
V3.0.0	+ Each port now defines portBASE_TYPE as the data type that is most efficient for that architecture. + The idle task is now created when the scheduler is started ...
V4.0.0	+ New co-routine functionality + Several kernel updates ...
V4.5.0	+ Added Mutex functionality. + Added the xQueueSendToFront(), xQueueSendToBack() and xQueuePeek() functionality ...
V4.7.0	+ Introduced the 'alternative' queue handling API + Introduced the counting semaphore macros ...
V4.8.0	+ Added new xQueueIsQueueEmptyFromISR(), xQueueIsQueueFullFromISR() ... API functions + Added new trace macros ...
V7.0.0	+ Introduced a new software timer implementation + Various enhancements to the kernel implementation in *tasks.c* ...
V8.0.0	+ Event groups + Centralised deferred interrupt processing ...
V8.2.0	+ Task notifications + Several kernel updates ...
V9.0.0	+ Tasks, semaphores, queues, timers and event groups can now be created using statically allocated memory + Added the xTaskAbortDelay() API function ...
V10.0.0	+ Stream buffers + Message buffers ...

decayed from 23.3% in V2.4.2 to 8.9% in V10.0.0. This change might be attributed to the growth rate of the support for various architecture ports, which is faster than that of the real-time scheduler functionality. However, it is notable that the real-time scheduler is crucial for FreeRTOS. We can see that the proportions of task function nodes in most versions account for more than 10%.

Table 4 Comparison of scheduler networks between FreeRTOS and Linux OS.

Network property	Average value (standard deviation)		Test result (p value)
	FreeRTOS	Linux OS	FreeRTOS vs Linux OS
C	0.021 (0.006)	**0.024** (0.004)	**Reject** (0.022)
$\langle k \rangle$	1.853 (0.135)	1.893 (0.139)	Do not reject (0.099)

Furthermore, when comparing the discrepancy between the real-time scheduler of FreeRTOS and a typical non-real-time scheduler of Linux OS, we build scheduler networks based on the source codes in the scheduler implementation files of FreeRTOS and Linux OS. It is noted that the scheduler of Linux OS is mainly implemented in the file /kernel/sched.c (or /kernel/sched/core.c starting from version 3.3) of the kernel source code. Obviously, these two scheduler networks are sub-networks of the FreeRTOS and Linux OS networks, respectively. To ensure the validity of results, we choose 51 major versions for Linux OS, ranging from versions 2.6.11 to 4.1, for which the range of release dates is close to that of the FreeRTOS versions. The null hypothesis is that for the two types of schedulers, the network property is sampled from the same distribution. In this study, we choose the clustering coefficient C and average degree $\langle k \rangle$ (i.e., average out-/in-degree) for comparison. The result is verified by means of the Wilcoxon-Mann-Whitney test.[42]

The comparison result is shown in Table 4. For a given criterion ($\alpha = 0.05$), after performing the test, we obtained a p value of 0.022 for the clustering coefficient, which means that the null hypothesis can be rejected at a 95% confidence, while for average degrees, the null hypothesis cannot be rejected. It can be observed that the average clustering coefficient for the Linux OS scheduler network is larger than that of the FreeRTOS scheduler network. This finding

illustrates that the interactions among the function implementations of the non-real-time scheduler of Linux OS are tighter than those of the real-time scheduler of FreeRTOS. The result is attributed to the non-real-time scheduler of Linux OS being more complex than the real-time scheduler of FreeRTOS, due to the more complex application domains of Linux OS. The real-time scheduler of FreeRTOS supports priority-based preemptive, cooperative, and hybrid operations.[36] Its scheduler algorithm is the highest priority first, while tasks of equal priority are executed in a round-robin fashion. Comparatively, for the Linux OS scheduler, the classifications of processes (e.g., interactive, batch, or real-time processes[43]) or the scheduler algorithms (e.g., completely fair scheduler[44]) are more complex. Therefore, it is reasonable that the non-real-time scheduler network of Linux OS has a larger clustering coefficient.

4. Conclusions

In this paper, we have performed an empirical investigation of the FreeRTOS real-time operating system, a commonly used RTOS for UAVs, in terms of evolution. We have collected 85 releases of FreeRTOS that range from V2.4.2 to V10.0.0, and conducted evolution analysis from a complex network perspective, concentrating on four research aspects: evolution of network properties, evolution of k-core structures, evaluation of functionality changes, and evolution of the real-time scheduler. Analytical results illustrate that the evolution of FreeRTOS well reflects the characteristics of small storage space support, various architectural designs, and high real-time requirement for embedded platforms.

There have been several interesting observations that could be useful for reliability analysis of UAV RTOSs. For example, the continuing growth of the size of FreeRTOS and the steady evolution of the graph coreness are well in accordance with Lehman's laws of software evolution. These network metrics can be utilized to measure the evolutions of software systems. In addition, the k-core analysis identifies the most crucial structure of FreeRTOS, which could instruct developers to pay more attentions to this region for testing UAV RTOSs. Moreover, the proposed metrics based on the k-core analysis could be utilized to evaluate major functionality changes of FreeRTOS, which have great potential to be adopted in other software systems for evaluation of functionality changes. A comparison of clustering coefficients between a real-time scheduler of FreeRTOS and a non-real-time scheduler of Linux OS demonstrates that the discrepancy between these two types of OS schedulers can be distinguished by utilizing the network metric.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 61772055) and Equipment Preliminary R&D Project of China (No. 41402020102).

References

1. Stankovic JA, Rajkumar R. Real-time operating systems. *Real-Time Syst* 2004;**28**(2):237–53.
2. Costa LDF, Oliveira Jr ON, Travieso G, Rodrigues FA, Villas Boas PR, Antiqueira L, et al. Analyzing and modeling real-world phenomena with complex networks: A survey of applications. *Adv Phys* 2011;**60**(3):329–412.
3. Erdös P, Rényi A. On random graphs I. *Publ Math Debrecen* 1959;**6**:290–7.
4. Merton RK. The Matthew effect in science. *Science* 1968;**159**(3810):56–63.
5. Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. *Nature* 1998;**393**(6684):440–2.
6. Barabási AL, Albert R. Emergence of scaling in random networks. *Science* 1999;**286**(5439):509–12.
7. Giot L, Bader JS, Brouwer C, Chaudhuri A, Kuang B, Li Y, et al. A protein interaction map of Drosophila melanogaster. *Science* 2003;**302**(5651):1727–36.
8. Cohen R, Erez K, Ben-Avraham D, Havlin S. Breakdown of the Internet under intentional attack. *Phys Rev Lett* 2001;**86**(16):3682.
9. Du WB, Liang BY, Yan G, Lordan O, Cao XB. Identifying vital edges in Chinese air route network via memetic algorithm. *Chinese J Aeronaut* 2017;**30**(1):330–6.
10. Saberi M, Mahmassani HS, Brockmann D, Hosseini A. A complex network perspective for characterizing urban travel demand patterns: Graph theoretical analysis of large-scale origin–destination demand networks. *Transportation* 2017;**44**(6):1383–402.
11. Wang WX, Wang BH, Yin CY, Xie YB, Zhou T. Traffic dynamics based on local routing protocol on a scale-free network. *Phys Rev E* 2006;**73**(2):026111.
12. Du WB, Zhou XL, Lordan O, Wang Z, Zhao C, Zhu YB. Analysis of the Chinese Airline Network as multi-layer networks. *Transport Res E-Log* 2016;**89**:108–16.
13. Lordan O, Sallan JM. Analyzing the multilevel structure of the European airport network. *Chinese J Aeronaut* 2017;**30**(2):554–60.
14. Du WB, Ying W, Yan G, Zhu YB, Cao XB. Heterogeneous strategy particle swarm optimization. *IEEE Trans Circuits-II* 2017;**64**(4):467–71.
15. Yan G, Vértes PE, Towlson EK, Chew YL, Walker DS, Schafer WR, et al. Network control principles predict neuron function in the Caenorhabditis elegans connectome. *Nature* 2017;**550**(7677):519.
16. Wang L, Fu YB, Chen MZ, Yang XH. Controllability robustness for scale-free networks based on nonlinear load-capacity. *Neurocomputing* 2017;**251**:99–105.
17. Louridas P, Spinellis D, Vlachos V. Power laws in software. *ACM Trans Softw Eng Meth* 2008;**18**(1):2.
18. Valverde S, Cancho RF, Sole RV. Scale-free networks from optimal design. *Europhys Lett* 2002;**60**(4):512.
19. Myers CR. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Phys Rev E* 2003;**68**(4):046116.
20. Sui YL, Xue JL. SVF: interprocedural static value-flow analysis in LLVM. *Proceedings of the 25th international conference on compiler construction*; 2016 Mar 17–18; Barcelona. New York: ACM; 2016. p. 265–6.
21. Sui YL, Xue JL. On-demand strong update analysis via value-flow refinement. *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*; 2016 Nov 13–18; Seattle. New York: ACM; 2016. p. 460–73.
22. Zheng XL, Zeng D, Li HQ, Wang FY. Analyzing open-source software systems as complex networks. *Physica A* 2008;**387**(24):6190–200.
23. Gao YC, Zheng Z, Qin FY. Analysis of Linux kernel as a complex network. *Chaos Soliton Fract* 2014;**69**:246–52.
24. Wang HQ, Chen Z, Xiao GP, Zheng Z. Network of networks in Linux operating system. *Physica A* 2016;**447**:520–6.
25. Xiao GP, Zheng Z, Wang HQ. Evolution of Linux operating system network. *Physica A* 2017;**466**:249–58.

26. Freertos.org [Internet]. London: Real Time Engineers Ltd.; [updated 2017 Nov 23; cited 2017 Nov 23]. Available from: https://www.freertos.org.
27. Stingu E, Lewis FL. A hardware platform for research in helicopter UAV control. In: Kimon PV, Paul O, Les AP, editors. *Unmanned aircraft systems*. Netherlands: Springer; 2008. p. 387–406.
28. Furci M, Casadei G, Naldi R, Sanfelice RG, Marconi L. An open-source architecture for control and coordination of a swarm of micro-quadrotors. *2015 international conference on uAircraft systems*; 2015 Jun 9–12; Denver. Piscataway: IEEE Press; 2015. p. 139–46.
29. Mistry J, Naylor M, Woodcock J. Adapting FreeRTOS for multicores: An experience report. *Software Pract Exper* 2014;**44**(9):1129–54.
30. Andrade WL, Machado PD, Alves EL, Almeida DR. Test case generation of embedded real-time systems with interruptions for FreeRTOS. *2009 12th brazilian symposium on formal methods*; 2009 Aug 19–21; Gramado. Berlin: Springer; 2009. p. 54–69.
31. Ferreira JF, Gherghina C, He GH, Qin SC, Chin WN. Automated verification of the FreeRTOS scheduler in Hip/Sleek. *Int J Softw Tools Technol* 2014;**16**(4):381–97.
32. Guan F, Peng L, Perneel L, Timmerman M. Open source FreeRTOS as a case study in real-time operating system evolution. *J Syst Software* 2016;**118**:19–35.
33. Lehman MM. Laws of software evolution revisited. *1996 5th European workshop on software process technology*; 1996 Oct 9–11; Nancy. Berlin: Springer; 1996. p. 108–24.
34. Seidman SB. Network structure and minimum degree. *Soc Networks* 1983;**5**(3):269–87.
35. Montresor A, De Pellegrini F, Miorandi D. Distributed *k*-core decomposition. *IEEE Trans Parall Distr* 2013;**24**(2):288–300.
36. Barry R. Mastering the FreeRTOS real time kernel-a hands on tutorial guide [Internet]. London: Real Time Engineers Ltd.; 2016 [cited 2017 Nov 23]. Available from: https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf.
37. Fagiolo G. Clustering in complex directed networks. *Phys Rev E* 2007;**76**(2):026107.
38. Batagelj V, Zaversnik M. An O(m) algorithm for cores decomposition of networks. *Comput Sci* 2003;**1**(6):34–7.
39. Zhang HH, Zhao H, Cai W, Liu J, Zhou WL. Using the k-core decomposition to analyze the static structure of large-scale software systems. *J Supercomput* 2010;**53**(2):352–69.
40. Zhang GQ, Zhang GQ, Yang QF, Cheng SQ, Zhou T. Evolution of the Internet and its cores. *New J Phys* 2008;**10**(12):123027.
41. Pan WF, Li B, Liu J, Ma YT, Hu B. Analyzing the structure of Java software systems by weighted K-core decomposition. *Futur Gener Comp Syst* 2018;**83**:431–44.
42. Sheskin DJ. *Handbook of parametric and nonparametric statistical procedures*. Boca Raton (FL): CRC Press; 2003.
43. Bovet DP, Cesati M. *Understanding the Linux Kernel: from I/O ports to process management*. Sebastopol (CA): O'Reilly Media; 2005.
44. Pabla CS. Completely fair scheduler. *Linux J* 2009;**2009**(184):4.