Journal of Information Hiding and Multimedia Signal Processing ©2017 ISSN 2073-4212
Ubiquitous International Volume **8**, Number **3**, May **2017**

Using Fault Infection and Propagation Mining Top Important Function Nodes in Software Execution Complex Network

Wanchang Jiang [1,2], Jiadong Ren [2], Yuan Huang [2]

[1]School of Information Engineering
Northeast Dianli University
Jilin, 132012, P. R. China

[2]College of Information Science and Engineering
Yanshan University
Qinhuangdao, 066004, P. R. China
jwchang84@163.com, jdren@ysu.edu.cn, 757918272@qq.com

Received November, 2016; revised February, 2017

Abstract. *Due to complicated relations of call and dependence between functions in software, failure of one function may propagate and infect other functions. Software execution complex network is constructed based on complex network to demonstrate internal complicated function call relations in software execution. Based on fault infection and propagation with relation in the software network, especially with direct relation in local network, a novel top-k important function nodes mining method is proposed to identify important and influential nodes that are susceptible to infection and make the range of fault wider and more serious. Firstly, a measure of fault infected degree (short as FID) is proposed to analyze the infection degree directly caused by called nodes to the node. On this basis, then a measure of fault infected index (FII) is designed to calculate how one node is infected by both its called nodes and itself. Functions with recursive call or mutual call are specially treated. Finally, with FII, fault infection capacity (FIC) is proposed to obtain the fault infection capacity caused by one node to other nodes in the network directly. Then a top-k important nodes mining algorithm is designed to mine top k important functions in the software network. Experimental results on open source software systems demonstrate that the method can reasonably mine top k important nodes in software network, especially in the upgraded versions.*

Keywords: Software complex network, Fault infection and propagation, Function calls, Top-k important nodes

1. Introduction.

The development of software techniques and increasing enhancement of function requirements make the software systems increasingly complicated. The fault of one entity may be propagated to other entities through the internal relation of the software. If some key entities in software are suffered by deliberate attacks, the failure may lead to collapse of the whole software system [1, 2]. Therefore, how to guarantee the reliability and stability of the software becomes more and more important.

Techniques of different granularity are proposed to improve the software reliability in the different phases of the software lifecycle. Fine-grained techniques of fault localization are proposed to identify the location of faults in software. Program spectra-based suspiciousness metrics Sokal and HAN are designed to compute suspiciousness value of each statement to be fault [3]. To improve the effectiveness of spectrum-based fault localization techniques, the ratio between non-violated and violated metamorphic test groups of test suites is investigated [4]. The spectra of fail test cases beyond balanced test suite are cloned to improve the performance of spectrum-based fault localization techniques [5]. A learning-based approach is proposed to combine multiple metrics for fault localization [6].

In order to make software testing and maintenance more targeted, the coarse-grained techniques are proposed to improve the reliability and stability of software. With features of divide-and-conquer, modularization, high intra-module cohesion, and low inter-module coupling, the software execution process is treated as a complex network [7, 8]. Based on weighted networks from dynamic software execution, critical modules or critical routes are used to improve testing efficiency and reduce testing cost [9]. With analyzing dynamic execution path in software, a novel approach is proposed to mine similar execution path to reduce software testing cases [10]. A new network growth model is developed to reproduce the particular features exhibited by software packages [11]. The fault of one function may be transmitted to other functions of software network, which eventually leads to the cascading failure [12]. To reduce the infection scope, In-degree metric is proposed to measure the importance of node in network [13]. The Laplacian-based centrality is extended to the case of general directed networks, and then arbitrary nodes can be quantitatively compared [14]. A graph-based characterization of a software system is used to capture its evolution and predict bug severity [15]. The relations of function calls are mapped to software execution networks, and node measurement score for measuring the importance of functions is defined [16].

However, on the one hand, the direct relation and cascade relation in network are not considered together in the above studies to measure fault infection and propagation. On the other hand, functions with recursive calls or mutual calls are ignored, which may lead to an accumulation of fault.

Therefore, software execution complex network is constructed to reflect the complicated calling relations of functions in the process of software execution. With the local and universal infection and propagation of one node, the degree of node being infected, the degree of node infecting other nodes and the degree of infecting itself are utilized, and then a measure of fault infection capacity FIC is proposed to measure the fault infection of the node to the network. Then the FIC-based top$k$ important nodes mining algorithm is designed to identify influential nodes in the network. At last, the experiment is conducted on three open source software systems, and top k important nodes in the software complex network are obtained. It will help developers and maintainers to understand the complicated software network, and then improve pertinence and effectiveness of maintenance for software system, especially for the upgraded versions.

The remainder of this paper is organized as follows. The constructing of software execution complex network is presented in Section 2. In Section 3, we propose fault infection measure FI of node in the software execution network. Section 4 presents the top important nodes mining algorithm. The experiments are designed on the typical software in Sections 5. At last, we conclude the work and discuss the future work in Section 6.

2. **Construction of software execution complex network.**

Since the method of modular and hierarchical design is used for software implementation, software consists of both entities (such as functions and modules) and calls and interdependence among entities. Software execution has the features of complex network, for example, it is not a random network or a regular network, the degree distribution obeys a power law form and so on. So the software execution complex network is built to reflect the complicated relations of entities in software execution.

Definition 2.1. *Software execution complex network. As the basic unit of software system, especially the process-oriented one, function is considered as the node, the relation of function calls between functions is considered as the directed edge, and the number of calls is considered as the weight of the corresponding edge. As a result, software execution complex network is constructed, and represented as a weighted directed network as G=(N, E, T). N represents the function node set* $\{n_i\}$*, where* n_i *is one of function nodes. E represents the relations of the edge set* $\{e_{ij}\}$*, wherein* $e_{ij} = (n_i, n_j)$ *is one edge made up of an ordered node pair. That is to say, node* n_i *directly calls node* n_j*, which is represented as* $n_i \rightarrow n_j$*. T represents the edge weight set* $\{t_{ij}\}$*, where* t_{ij} *is the number of times that* n_i *directly calls node* n_j*.*

Definition 2.2. *Local network of node* n_k*. Local network of node* n_k *is the weakly connected sub-graph containing node* n_k *in software execution complex network, which consists of all nodes that can be reached the node* n_k *and all nodes that node* n_k *can reach. For* $\forall$ *node* n_j*, if the network has a path of directed edges from node* n_k *to node* n_j*, all nodes (including node* n_j*) in the path is added into the local network of node* n_k*. For* $\forall$ *node* n_i*, if the network has a path of directed edges from node* n_i *to node* n_k*, all nodes (including node* n_i*) in the path is added into the local network.*

Take software package *cflow*-1.3 in the open source software library [17] as an example, relations of function calls will be obtained when *cflow*-1.3 performs an analysis task of C language code. And then the software execution complex network of *cflow*-1.3 is constructed as a weighted directed network, shown in Figure 1 as follows.

As shown in the figure, the local network of node *linked_list_append* is included in the figure on the lower right, function node *linked_list_append* calls node *deref_linked_list*, and it is called by function nodes of *add_name*, *append_symbol*, *reference*, *add_reference* and *call*. With relations of function calls in the software network, if one function entity has fault, then the fault may be propagated to other functions and even infect other functions. As a result, we propose an importance measure of nodes in the software network based on both local and universal fault infection and propagation. The important and influential nodes involved in the fault infection and propagation are closely related to the software stability and robustness.

3. **Fault infection capacity measure of function nodes in software network.**

The fault of one node may be propagated to other nodes in the software network with different infection capacity. If the weighted directed network G = (N, E, T) has a path of directed edges from node n_k to n_j, the fault of n_j may be propagated to n_k. In other word, node n_k can be infected by node n_j through the path of edges.

Due to the specificity of software network, each function node in the software network may realize its functionality with help of called nodes, especially directly called ones. The fault mostly is propagated to a limited range of nodes in the network, that is, local network of the node. Therefore, if there is a directed edge $e \in$ E from node n_i to node n_j and n_i, $n_j \in$ N, then the fault of n_i may be infected by n_j. A measure named fault infected degree is proposed to analyze the degree of infection to the node directly caused by the called nodes.

Fault infected degree (FID). Node n_j directly calls other nodes $\{n_{j_k}\}$ directly, $\{n_{j_k}\}$ is called nodes set of node n_j, the number of elements of the set is N_j^d. When one or some of these called nodes have fault, the fault can be propagated to the node n_j through the calling relation, that is node n_j may be infected. To measure the possibility of the fault of the directly called nodes turning their calling node n_j into fault, a measure named fault

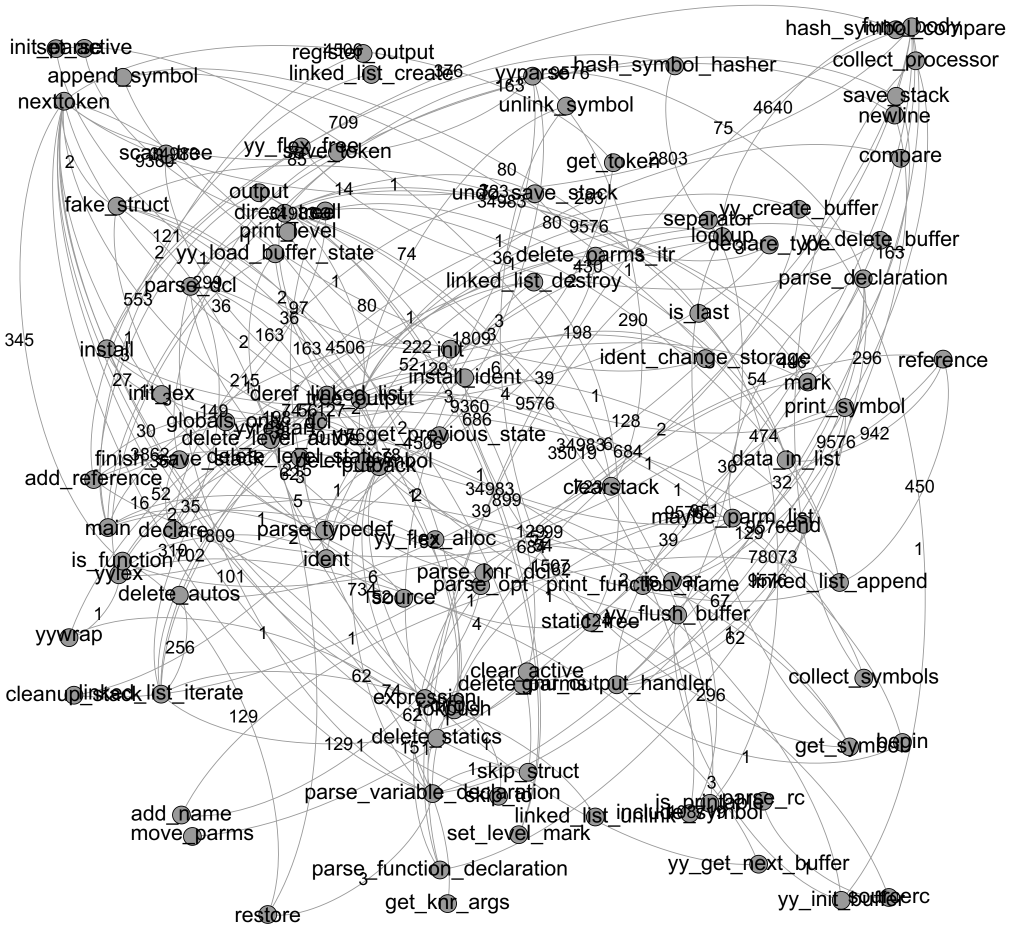

FIGURE 1. Construction of software execution complex network of *cflow*-1.3.

infected degree is proposed, short as FID.

$$\mathrm{FID}(n_j) = \sum_{k=1}^{N_j^d} w_{j,j_k} * \mathrm{FII}(n_{j_k}) \tag{1}$$

Where $\mathrm{FII}(n_{j_k})$ is fault infected index (FII) of the node n_{j_k}, and the fault infection caused by the node itself and other nodes is reflected. To reflect different importance of the called nodes in causing infection to the node n_j, weight w_{j,j_k} of node n_{j_k} is calculated by using the edge weight t_{j,j_k} of the software network as follows.

$$w_{j,j_k} = t_{j,j_k} / \sum_{h=1}^{N_j^d} t_{j,j_h} \tag{2}$$

Then the definition of fault infected index FII is given to measure fault infection caused by the node itself and other nodes in the software network.

Fault infected index (FII). Node n_k may fault with failure in the node itself. Furthermore, it also may be infected by its called nodes. To measure how node n_k infected by both its called nodes and itself, the corresponding possibility of the node to be fault is defined as fault infected index, $\mathrm{FII}(n_k)$.

$$\mathrm{FII}(n_k) = d * \mathrm{FID}(n_k) + (1 - d) * \mathrm{FIID}(n_k) \tag{3}$$

where $\mathrm{FID}(n_k)$ is the fault infected degree of called nodes of n_k, contains the infection information of indirectly called nodes of n_k. The local infection and propagation of one node is considered in calculation the fault infected index. And, $\mathrm{FIID}(n_k)$ is fault infected itself degree, which is defined to obtain the degree of fault infected by itself. With the adjustable factor d, the role $\mathrm{FID}(n_k)$ and $\mathrm{FIID}(n_k)$ in calculating $\mathrm{FII}(n_k)$ is balanced. Only $\mathrm{FIID}(n_k)$ is used to calculate $\mathrm{FII}(n_k)$, when one node has no called nodes, that is, leaf node with In-degree being zero.

Furthermore, there are some special nodes in the software network with complicated function calls, such as the node with function recursive call and the two nodes with function mutual call. These kinds of function call should be processed to avoid calculation of endless loop, and the calculation is as follows.

$$\mathrm{FII}(n_k) = d * \mathrm{FIID}(n_k) \tag{4}$$

$$\mathrm{FII}(n_k) = \mathrm{FIID}(n_k) + \mathrm{FIID}(n_i) \tag{5}$$

Finally, some nodes call node n_j in the software network. Once fault occurs in node n_j, the fault can be propagated to these nodes. A measure of fault infection capacity (FIC) of node is proposed to measure the direct fault infection capacity of the node to the network.

Fault infection capacity (FIC). Fault infection capacity is proposed to measure the fault infection capacity of one node to its corresponding calling nodes in the network, namely, the fault infection directly caused by the node to the network. The formula of $\mathrm{FI}(n_j)$ is shown as follows.

$$\mathrm{FIC}(n_j) = \sum_{m=1}^{N_j^c} u_{j_m,j} * \mathrm{FII}(n_j) \tag{6}$$

The fault of node n_j can be propagated to nodes that call it through the directed edge, the degree of the calling nodes to be infected is different. Therefore weight $u_{j_m,j}$ is given to reflect the weight of infection to each calling node, which can be obtained by using edge weight $t_{j_m,j}$ of software network with help of formula 7).

$$u_{j_m,j} = t_{j_m,j} / \sum_{h=1}^{N_j^c} t_{j_m,j} \tag{7}$$

And fault infection capacity FIC is used to measure the direct infection influence of fault of node to the software network.

4. **Top k important nodes of software network mining algorithm FIC_TINSNM.**

We propose a top-k important nodes mining algorithm to mine important nodes (especially top ones) of software network through application of the measure of fault infection capacity FIC, short as FIC_TINSNM. The algorithm FIC_TINSNM has three phases. Firstly, with the constructed weighted directed software execution complex network, measures $\mathrm{FI}(n_j)$, $\mathrm{FII}(n_j)$, $\mathrm{FID}(n_j)$, $\mathrm{FIID}(n_j)$ and etc. of each node are initialized. Second, all nodes that can arrive from node n_j through one directed edge are traversed, FID of n_j is calculated. Then all nodes that can reach node n_j through one directed edge are traversed, FIC of node n_j is calculated. At last, FIC is obtained as the measure of influence and importance of nodes in the network. As a result, top k important nodes are identified.

The detailed steps of the top-k important nodes mining algorithm FIC_TINSNM are shown as follows.

In the top important nodes mining algorithm, the cascade relation is considered in the process of calculating fault infection capacity FIC of each node, especially the direct relation of local network of one node. Besides the information on fault caused by the node itself(FIID in line 33), the relation of function calls between the node and its called nodes (FID in line 29), and the relation of calls between its calling nodes and the node (FIC in line 39) are also considered to obtain fault infection capacity FIC of each node. To solve the problem of functions with recursive calls or mutual calls, lines 21 and 25 in the algorithm are designed respectively. As a result, based on local and universal infection and propagation, the obtained top k important nodes may contain nodes ignored by other methods.

5. **Experiment.**

5.1. **Experiment setup.**

To illustrate the reasonability of the FIC-based top-k important nodes of software network mining algorithm FIC_TINSNM, the performance with other three measures of In-Degree, Degree and PageRank is compared. Three software systems in the open source software library [17] are selected in the experiment, that is, software *gzip* (for file compression), *tar* (for bundling more files into a single one) and *cflow* (for analysis of a collection of C source files) respectively. And four versions of program package of each software system are used.

The test source of *instrument.c* is used to track relations of function calls when each version of *gzip*, *tar*, *cflow* is executed. The tool of *Pvtrace* to analyze the tracked file and generate document that record relations of function calls. Then with the program realized by Java, relations of function calls are mapped to a weighted directed network. That is, the software execution complex network is obtained. The top-k important nodes mining algorithm is used to obtain top k important function nodes in the software network. The experiments are carried out under the environment of Linux system.

5.2. **Experiment results.**

With the top-k important nodes mining algorithm FIC_TINSNM, FIC of each node in the software network is calculated firstly. With the number of nodes in corresponding intervals of FIC, the FIC distribution of software network of *cflow*, *gzip* and *tar* is shown in Figure 2, 3 and 4 respectively.

As shown in Figure 2, 3 and 4, the number of nodes versus FIC features unevenly distribution, the overall distribution curve of FIC shows the number of corresponding nodes decreases with the value of FIC. For example, FIC mostly distributes in the range of small FIC of versions of software *cflow*, only a few nodes have high FI value. Therefore, these nodes of software network can be distinguished from other ones in terms of FIC. Comparing FIC of different versions of the software, we can find that FIC distribution varies a little in process of upgrading software version.

By using FIC measure method, function nodes of software networks of *gzip*, *tar* and *cflow* are ranked. The top 10 important nodes of each software network are listed in Table 1, 2 and 3 respectively.

As shown in Table 1, ten functions with top FIC are slightly different in software execution complex network of each version. With the maximum FIC, function *nexttoken* is the most important node in network of each version, and so as the next important function node *gnu_output_handler*.

Functions of *linked_list_append*, *linked_list_iterate* and *linked_list_destroy* in software network of *cflow*-1.3 and *cflow*-1.4 are new top functions that do not emerge in that of versions 1.1 and 1.2. These new top important function nodes offer reliable

```
Algorithm:1
Input: software execution complex network G=(N, E, T)
Output: top k important nodes N_{top-k}={$n_{t_m}$}
1. Initialize FIC($n_j$) and FID($n_j$) of each node with 0.
2. Initialize FII($n_j$) of each node with 0, and FII($n_j$) with $1/|N|$.
3. Calculate in-degree and out-degree of each node
4. For each node
5.   If(out-degree($n_i$)==0)
6.     FII($n_i$)=FIID($n_i$)
7.   End If
8. End For
9. For each node
10.   If(out-degree($n_i$)==0 and in-degree($n_i$)==0)
11.    FII($n_i$)=FIID($n_i$)
12.    FIC($n_i$)=FII($n_i$)
13.   End If
14.End For
15.While($\exists n_m$, FIC($n_m$)==0)
16.   For($i$=1;$i <= n$;$i$++)
17.    $in\_num$=0
18.    If(FID($n_i$)==0)
19.      If($rel[i,i] \neq$0)
20.        $in\_num$++;
21.        FII($n_i$)=$d$*FIID($n_i$)
22.      End If
23.      For($j$=1;$j <= n$;$j$++)
24.       If($rel[i,j] == rel[j,i]$)
25.         FII($n_j$)= FIID($n_j$)+ FIID($n_i$);
26.       End If
27.      End For
28.      If($in\_num$==outDeg($n_i$))
29.       FID($n_i$) $= \sum_{k=1}^{N_i^d} w_{i,i_k} *$ FII($n_{i_k}$)
30.      End If
31.     End For
32.     If(FII($n_i$)==0 and FID($n_i$) $\neq$0)
33.      FII($n_i$)= $d$*FID($n_i$)+(1-$d$)*FIID($n_i$)
34.     End If
35.     If(inDeg($n_i$)==0)
36.      FIC($n_i$)=FII($n_i$)
37.     End If
38.     If(FII($n_i$) $\neq$0 and FID($n_i$) $\neq$0)
39.      FIC($n_i$) $= \sum_{m=1}^{N_i^c} u_{i_m,i} *$ FII($n_i$)
40.     End If
41.   End For
42.End While
43.Rank nodes on the basis of the measure FIC
44.Output top k important nodes N_{top-k}={$n_{t_m}$} in the network
```

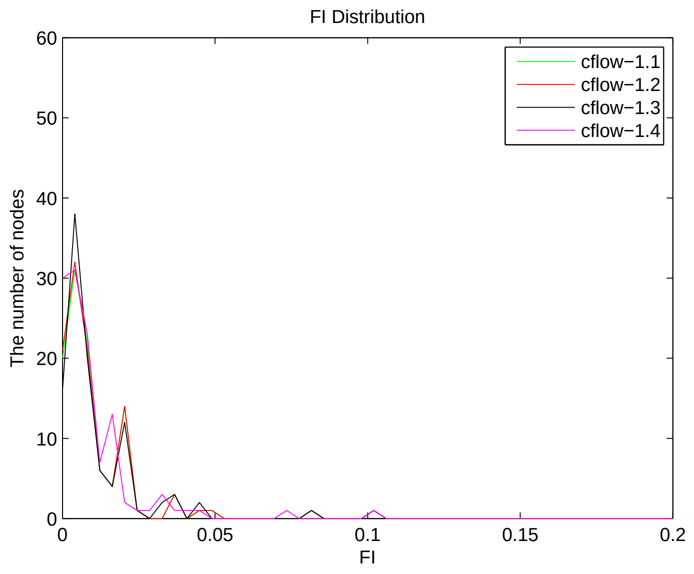

FIGURE 2. FIC distribution of software network of each version of $cflow$.

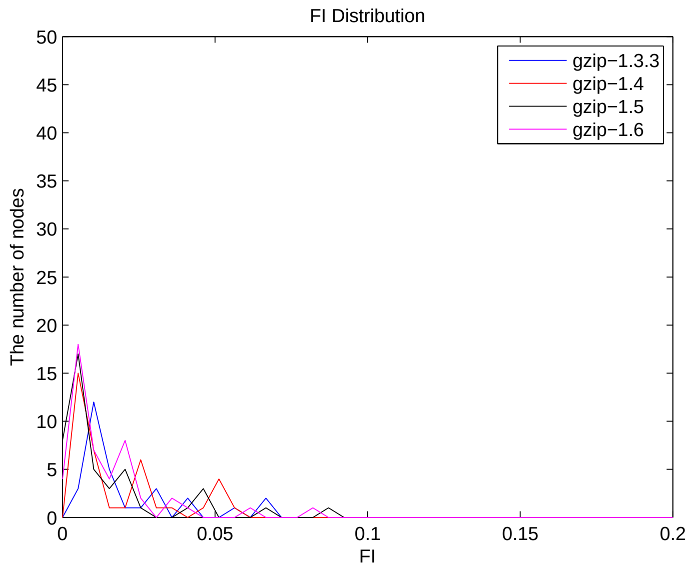

FIGURE 3. FIC distribution of software network of each version of $gzip$.

guarantee for maintenance of the upgraded software versions. *linked_list_append* calls *deref_linked_list*, and it is called by functions of *add_name*, *append_symbol*, *reference*, *add_reference* and *call*. Compare with the low versions, *append_symbol* also is a new function in version 1.3 as a called function of *linked_list_append*, which appends the symbol to the tail of the table entry. If function *linked_list_append* faults, the fault may be propagated to its five calling nodes. Function *linked_list_iterate* has the bigger FIC value than function *linked_list_destroy* in $cflow$-1.3, and *linked_list_iterate* calls function *linked_list_destroy*. The fault of *linked_list_destroy* may be propagated to its calling node of *linked_list_iterate*. The above three new functions should be paid more attention in maintenance of versions in the future.

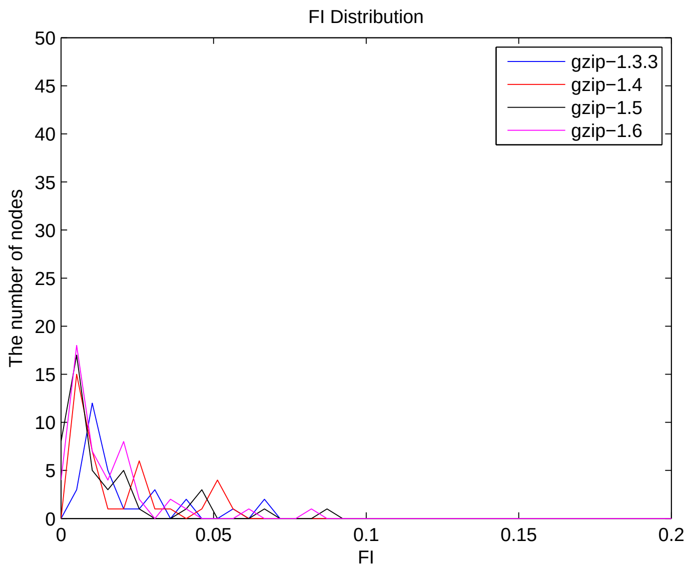

FIGURE 4. FIC distribution of software network of each version of *tar*.

TABLE 1. Top 10 important nodes ranking of software executing network of *cflow*

	cflow-1.1	*cflow*-1.2	*cflow*-1.3	*cflow*-1.4
1	*nexttoken*	*nexttoken*	*nexttoken*	*nexttoken*
2	*gnu_output_handler*	*gnu_output_handler*	*gnu_output_handler*	*gnu_output_handler*
3	*install*	*install*	*linked_list_append*	*linked_list_append*
4	*lookup*	*lookup*	*lookup*	*lookup*
5	*append_to_list*	*append_to_list*	*yyrestart*	*delete_symbol*
6	*yyrestart*	*yyrestart*	*linked_list_iterate*	*yyrestart*
7	*delete_symbol*	*delete_symbol*	*delete_symbol*	*linked_list_iterate*
8	*parse_dcl*	*parse_dcl*	*install*	*install*
9	*scan_tree*	*scan_tree*	*linked_list_destroy*	*linked_list_destroy*
10	*is_last*	*is_last*	*parse_dcl*	*parse_dcl*

TABLE 2. Top 10 important nodes ranking of software executing network of *gzip*

	gzip-1.3.3	*gzip*-1.4	*gzip*-1.5	*gzip*-1.6
1	*main*	*main*	*file_read*	*read_buffer*
2	*flush_block*	*read_buffer*	*send_bits*	*main*
3	*bi_reverse*	*flush_block*	*main*	*copy_block*
4	*build_tree*	*write_buf*	*get_suffix*	*file_read*
5	*send_bits*	*bi_reverse*	*write_buf*	*build_tree*
6	*do_stat*	*build_tree*	*bi_reverse*	*bi_reverse*
7	*file_read*	*send_bits*	*build_tree*	*fstat*
8	*write_buf*	*open_and_stat*	*open_and_stat*	*_moddi3*
9	*gen_codes*	*fstat64*	*strlwr*	*strcpy*
10	*scan_tree*	*file_read*	*write_buffer*	*ct_tally*

As shown in Table 2 and 3, function *main* play an important role in *gzip* and *tar*. Compare with other function nodes, sometimes it can be ignored for simplifying maintenance.

Table 3. Top 10 important nodes ranking of software executing network of *tar*

	tar-1.13	*tar*-1.20	*tar*-1.21	*tar*-1.27
1	*from_oct*	*from_header*	*from_header*	*from_header*
2	*set_stat*	*set_stat*	*set_stat*	*main*
3	*main*	*main*	*main*	*page_aligned_alloc*
4	*flush_archive*	*page_aligned_alloc*	*page_aligned_alloc*	*flush_archive*
5	*find_next_block*	*flush_archive*	*flush_archive*	*flush_read*
6	*name_next*	*flush_read*	*flush_read*	*gnu_flush_read*
7	*child_open_for_uncompress*	*gnu_flush_read*	*gnu_flush_read*	*_gnu_flush_read*
8	*xclose*	*_gnu_flush_read*	*_gnu_flush_read*	*tar_sparse_init*
9	*flush_read*	*read_header_primitive*	*read_header_primitive*	*transform_member_name*
10	*quote_copy_string*	*tar_sparse_init*	*tar_sparse_init*	*transform_name_fp*

To further illustrate the rationality of the result of top k important nodes mining based on FIC, four function ranking strategies of In-Degree, Degree, PageRank and FIC are utilized respectively to rank nodes of software network. The top 10 important functions ranking based on In-Degree, Degree, PageRank and FIC is listed respectively in Table 4.

Table 4. Top 10 important nodes ranking of software executing network of *tar* with four different methods

In Degree	Function	Degree	Function	Page Rank	Function	FIC	Function
12	*nexttoken*	14	*nexttoken*	0.529	*scan_tree*	0.102	*nexttoken*
8	*putback*	11	*direct_tree*	0.018	*lookup*	0.081	*gnu_output_handler*
5	*linked_list_append*	10	*parse_variable_declaration*	0.015	*print_symbol*	0.046	*linked_list_append*
5	*lookup*	9	*tree_output*	0.015	*hash_symbol_compare*	0.045	*lookup*
5	*gnu_output_handler*	8	*putback*	0.015	*nexttoken*	0.037	*yyrestart*
4	*mark*	7	*main*	0.013	*gnu_output_handler*	0.037	*linked_list_iterate*
3	*install*	7	*fake_struct*	0.012	*hash_symbol_hasher*	0.037	*delete_symbol*
3	*yy_load_buffer_state*	7	*lookup*	0.012	*ident*	0.034	*install*
3	*tokpush*	7	*parse_dcl*	0.011	*deref_linked_list*	0.032	*linked_list_destroy*
3	*restore*	7	*linked_list_iterate*	0.011	*static_free*	0.024	*parse_dcl*

As the results shown in the table, the measure method of In-degree may be ineffective in distinguishing nodes. However, the measure of FIC can distinguish each other effectively.

Take *cflow*-1.3 as an example, there are 8 nodes with In-degree value being 3, such as functions of *install*, *linked_list_destroy* and *include_symbol*. In contrast, these nodes could be ranked and mined by the algorithm FIC_TINSNM. Furthermore, there are 25 nodes with In-degree value being 2. The weakness of PageRank method is exposed. Fucntion *scan_tree* is the first ranking function for the recursive call relation, and thus it has the obvious larger PageRank value. And since fucntion *print_symbol* has a mutual call relation with *gnu_output_handler*, it has a top ranking with method of PageRank.

When *cflow*-1.3 is executed with a task of analysis, if function of *linked_list_append* has fault, the failure may be transmitted to other functions directly or indirectly. Its infection to directly calling nodes is considered, which will increase the probability of occurring fault in the network. It may lead at most 22.6% of functions in the software network to be fault, and 15.1% for *linked_list_iterate*. Whats more, the direct infection is paid more attention in the process of calculating FID and FII to mine important nodes, and the indirect infection is also considered. However, the direct and cascade relations of functions are not considered adequately in other measures, and some important functions are ignored.

6. Conclusions.

A good understanding of nodes of software systems is really important for improving software stability and robustness in the process of upgrading software version. The failure of function may infect other functions with function calls. With reference to complex network, software execution complex network is constructed to reflect the complicated relation of function calls in software execution. To measure the importance of nodes in the network, the measure of fault infection capacity FIC is proposed to measure the fault infection caused by the node to the network. With fault infection and propagation, the direct relation of one node is mainly considered, and the cascade relation is also considered to calculate the fault infection capacity. With the measure of FIC, then the top-k important nodes mining algorithm is designed to obtain the most important and influential function nodes in the network. Experiment results on open source software shown that top important nodes in software network can be obtained by the algorithm.

In the future work, besides relations of function calls, relation of multi-granularity in software network should be emphasized to design the measure of the influence of each node on the network in order to completely master the internal structure and execution of software systems.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No.61572420), the Natural Science Foundation of Hebei Province (No.F2014 203152) and the Education Department of Jilin Province [2016]94.

REFERENCES

[1] E. Zio, L. R. Golea, G. Sansavini, Optimizing protections against cascades in network systems: A modified binary differential evolution algorithm, *Reliability Engineering & System Safety*, 2012, 103, pp. 72-83.

[2] S. Chen, X. Zou ,L. Hui, Q. Xu, Research on robustness of interdependent network for suppressing cascading failure, *Acta Physica Sinica*, 2014,vol.63, no.2, pp. 257-264.

[3] L. Naish, H. J. Lee and K. Ramamohanarao, A model for spectra-based software diagnosis, *ACM Trans on Software Engineering and Methodology*, 2011, vol.20, no.3, pp.1-32.

[4] P. Rao, Z. Zheng, T.Y. Chen, N. Wang and K. Cai, Impacts of Test Suite's Class Imbalance on Spectrum-Based Fault Localization Techniques, *Proc. of the 13th International Conference on Quality Software*, Najing, China,2013, pp.260-267.

[5] P. DanielK.Y. Sim,S. Seol, Improving spectrum-based fault-localization through spectra cloning for fail test cases, *Contemporary Engineering Sciences*, 2014, vol.7, no.14, pp.677-682.

[6] J. Xuan, M. Monperrus, Learning to combine multiple ranking metrics for fault localization, *Proc. of Int'l Conf. on Software Maintenance and Evolution*, 2014, Victoria, Canada, pp.191-200.

[7] K. Cai, B. Yin, Software execution processes as an evolving complex network, *Information Sciences*, 2009,vol.179, no.12, pp.1903-1928.

[8] Lovro ?ubelj, Marko Bajec. Software Systems through Complex Networks Science: Review, Analysis and Applications, *Proceedings of the First International Workshop on Software Mining*, Beijing, China, 2012, pp. 6-18.

[9] K. Zhou, W. Lan, J. Feng, Software execution process as weighted complex networks, *Computer Engineering and Applications*, 2011, vol.47, no.17, pp.51-55.

[10] J. Ma, D. Zeng, H. Zhao, Modeling the growth of complex software function dependency networks, *Information Systems Frontiers*, 2012, vol.14, no.2, pp.301-315.

[11] J. Wang, Y. Liu, F. Mei , C. Zhang, Modeling cascading failures for Internet based on congestion effects, *Journal of Computer Research and Development*, 2010, vol.47, no.5, pp.772-779.

[12] X. Wang, Complex network: Topology, dynamics and synchronization, *International Journal of Bifurcation and Chaos*, 2002, vol.12, no.5, pp.885-916.

[13] N. Masuda, H. Kori, Dynamics-based centrality for directed networks, *Physical Review E-Statistical, Nonlinear, and Soft Matter Physics*, 2010, 82(5 Pt 2):056107,pp.1-11.

[14] H. He, Y. Liu; J. Dong, et al, A novel approach to mine software similar execution paths based on node rank, *ICIC Express Letters*, 2016,vol.10, no.2, pp.323-330.

[15] P. Bhattacharya, M. Iliofotou, I. Neamtiu, et al, Graph-based analysis and prediction for software evolution, *Proceedings of International Conference on Software Engineering*, Zurich, Switzerland, 2012, pp.419-429.

[16] H. He, J. Wang, J. Ren, Measuring the importance of functions in software execution network based on complex network, *International Journal of Innovative Computing, Information and Control*, 2015, vol.11, no.2, pp.719-731.

[17] http://sourceforge.net/