复杂软件的级联故障建模

王 健[1),2)] 刘衍珩[1),2)] 刘雪莲[3)]

[1)](吉林大学计算机科学与技术学院 长春 130012)

[2)](吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

[3)](吉林大学软件学院 长春 130012)

摘 要 软件复杂性的增加直接导致软件故障的复杂性增加.从系统科学的角度出发,采用复杂网络等方法和技术在整体上研究复杂软件的级联故障特性是研究软件质量的一个新视角.以函数为节点、以调用关系为边,提出了使用有向边和边权表述函数间的调用关系和紧密程度的拓扑模型.通过引入函数容错能力和软件故障强度建立复杂软件的级联故障模型,模拟软件运行时的故障传播行为.对 3 个实际软件网络的实验结果表明,弱的故障强度、少的初始故障节点和高的容错能力均会减缓故障的传播速度和缩小故障的波及范围.分析了部分软件节点最终未被感染的原因,文中模型有助于为软件工程人员准确检测复杂软件的复杂性和容错性提供新的思路.

关键词 复杂网络;软件复杂性;软件容错性;级联故障;软件度量

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.01137

Model for Cascading Faults in Complex Software

WANG Jian[1),2)] LIU Yan-Heng[1),2)] LIU Xue-Lian[3)]

[1)](*College of Computer Science and Technology, Jilin University, Changchun* 130012)

[2)](*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun* 130012)

[3)](*College of Computer Science and Technology, Jilin University, Changchun* 130012)

Abstract The increasing of software complexity directly results in the increased complexity of software faults. From the perspective of system science, using complex networks theory to modeling cascading faults in complex software provides a new approach to evaluate software quality. Functions and call relations are extracted into nodes and edges, respectively. Directional edges and corresponding weight values are used to express call relations and tightness degree. By introducing two concepts of function fault-tolerant capability and software fault intensity, a cascading fault model is proposed to explore fault propagation behaviors. Simulations on three practical software networks show that a weak fault intensity, a few number of initial faults, and a strong fault-tolerant capability can slow down the spreading speed and thus reduce the affect range. Some reasons for a few nodes that are not disrupted by faults are analyzed. The model is expected to give engineers a new approach to correctly evaluate the complexity and the fault-tolerant capability of complex software.

Keywords complex networks; software complexity; software fault tolerant; cascading fault; software metric

---

收稿日期:2010-09-12;最终修改稿收到日期:2011-04-27.本课题得到国家自然科学基金(60973136,61073164)、科技部国际科技合作与交流专项项目(2008DFA12140)、欧盟合作项目(155776-EM-1-2009-1-IT-ERAMUNDUS-ECW-L12)、吉林大学研究生"985 工程"创新计划项目(20101029)、吉林大学基本科研业务费项目(201103136)资助.**王 健**,男,1981 年生,博士研究生,讲师,主要研究方向为复杂网络和网络安全.E-mail: wangjian591@gmail.com. **刘衍珩**(通信作者),男,1958 年生,博士,教授,博士生导师,主要研究领域为网络安全、移动 IP 和网络服务质量.E-mail: lyh_lb_lk@yahoo.com.cn. **刘雪莲**,女,1984 年生,硕士研究生,主要研究方向为可信软件.

1 引 言

随着人们对软件功能的需求越来越高，软件结构也日益复杂，函数与函数、模块与模块、系统与系统等实体之间的互联互通与相互作用使软件整体结构表现出一些全新的特性，研究人员陆续发现软件系统的拓扑结构可以抽象成网络结构，并且具有小世界和无标度等复杂网络特性. 由于动态性和开放性的增加，复杂软件时刻受到各种故障和攻击的威胁. 例如，1996 年 6 月 4 日，阿丽亚娜 5 型火箭由于整型加速度值产生溢出，造成以加速度为参数的速度、位置等变量计算错误，程序只得进入异常处理模块，引爆自毁；2009 年 9 月 12 日，MSN 美国总部服务器瘫痪超过 1 小时，导致全球 900 万用户受到影响；2010 年 1 月初，由于一个软件兼容性故障导致美国约有 8000 至 10000 台 GPS 接收机失效.

图 1 给出了故障在 SNLC①(作者单位用于编译原理教学的一个简单但功能完整的 SNL 语言编译器)传播的情况. 对于给定的输入目标文件，首先记录在整个执行过程中每个函数在每个时刻的正确输出值，然后在 SNLC 源代码中随机地挑选一定数量的函数注入人为错误，对比正常函数在后续的执行过程中的输出值与之前的正确输出值，以此观察故障的传播特性. 图 1 的横坐标为以函数为单位的执行步骤，纵坐标为发生故障的函数数目. 从图 1 中可以看出，如果一个或者少数几个函数发生故障，该故障可能会随着调用和依存关系传播至其它函数而引发其它函数无法正常运行，最终导致部分或者整个系统的崩溃，称之为“级联故障”[1].

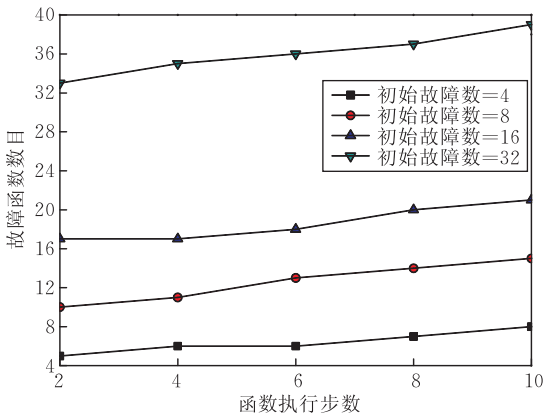


图 1 在 SNLC 中的故障传播实例

复杂软件作为典型的复杂系统可以运用复杂网络理论进行建模和分析，但是，目前却很少有人直接从软件工程的角度系统地研究复杂软件的质量特征. 因此，有必要从复杂网络的视角，结合软件工程实践，对复杂软件的级联故障特性进行深入研究.

在软件工程界，多种软件质量度量方法已经被提出，但大部分是建立在分析源代码的基础上. 2002 年，Valverde 等人[2]首先将软件结构抽象为网络拓扑，发现软件网络具有“小世界”和“无标度”现象，符合复杂网络的特点. 此后很多研究人员使用复杂网络理论建模和分析软件内部结构，借助复杂网络的数学理论和方法分析软件网络的各种统计特性，对软件进行质量度量及优化. 2004 年，美国卡内基·梅隆大学软件工程研究所发布了复杂巨系统软件专项调研，试图解决复杂巨系统的软件工程问题[3]. 2005 年，13 个跨国 IT 公司(包括 IBM、HP、Nokia 等)宣布合作研制网络化软件. 2006 年，我国学者何克清和李德毅等人将复杂网络理论和方法引入到软件工程的设计之中[4-5]. Cai 和 Yin[6]将软件系统的执行过程视为一个演化的有向复杂网络结构，引入了软件镜像图概念，并发现虽然在拓扑层面上软件执行过程表现出小世界现象，但在时间角度上不再体现小世界特征，其度分布可以表示为幂律分布也可以表示为指数函数. 目前，基于复杂网络理论对软件系统的研究工作大多集中在从不同方面和粒度对各类软件系统的复杂特征进行分析，揭示结构性特性，缺少从复杂网络角度研究软件质量特征的工作. 另外，目前常用的软件网络模型[7]虽然能够表示软件函数间的相互调用关系，但是无法表示该关系的紧密程度. 事实上，各函数被调用次数的差异从复杂网络的角度可以被认作是节点间相互作用的差异. 为了表示这种紧密程度，本文提出了一种具有边权的软件网络模型，用以表征函数调用关系和紧密程度；并在此基础上引入了软件故障强度和函数容错能力，设计了一种级联故障模型；同时，对实际软件网络、随机网络和无标度网络的大量实验揭示了故障触发方式、故障强度、初始故障数量、容错能力等对级联故障的影响，并分析了其中的原因，讨论了部分软件网络节点最终未被感染的原因.

本文第 2 节介绍软件复杂性和容错性的相关工作；第 3 节详细地给出软件网络模型和级联故障模型；第 4 节通过大量实验对比分析级联故障在实际软件网络中的传播情况以及各模型参数对级联故障的影响，并给出部分软件网络节点最终未被感染的原因；最后总结全文.

① SNLC. http://www.hackchina.com/en/cont/136801, Aug, 2009

2　相关工作

2.1　软件复杂性

在软件工程领域，已有多种方式对软件复杂性进行度量. 代码行数度量法以程序的总代码行数作为程序复杂性的度量值，但比较粗糙，实际中也很少应用. McCabe 度量法[8]以图论为工具，先画出程序图，然后用该图的环路数作为程序复杂性的度量，但其实质上是对程序控制流复杂性的度量，并不考虑数据流，因而其科学性和严密性有一定的局限. Halstead 度量法[9]通过计算程序中的运算符和操作数的数量对程序的复杂性加以度量，但只考虑了程序的数据流而没有考虑控制流，因而也不能从根本上反映程序的复杂性.

复杂网络的出现，为描述软件系统的复杂性提供了一个新的途径. 复杂软件系统的拓扑结构可以从服务、构件、模块、包、接口、类、函数等不同粒度进行网络拓扑建模，将上述元素视为组成网络结构的节点，这些元素之间的作用关系如调用关系、继承关系、聚合关系、包含关系、协作关系等视为边. 目前，相关研究人员主要是通过复杂网络的理论和方法对软件系统进行研究，研究对象主要是开源软件，采用逆向工程等方法，从开源软件源代码中抽取出软件类图，再建立软件网络模型进行分析，进而发现各种统计特性，并针对研究现象得出一些客观规律.

Valverde 等人[2]使用复杂网络方法描述软件拓扑结构，将面向对象软件的类图作为研究对象，提取类之间的继承和关联关系作为边，构造了软件结构的复杂网络拓扑图. Alessandro 等人[10]提出了一种分析 C 语言源程序的方法，并分析了 Linux kernel 等开源软件，忽略来自外部的库文件. 此后，国内外学者对大量的开源软件进行了分析，进一步验证了软件结构的复杂网络现象[11-13]. 随着研究的深入，研究人员认为无向图不能确切表示诸如类的继承、调用等关系，转而使用有向图进行建模[14-16]. 本文提出一种使用加权有向图建模软件结构的方法，能够更加准确地描述软件内部实体之间的依赖关系.

2.2　软件容错性

软件的容错性是指故障出现后，软件系统通过自身调节不致崩溃且仍能正常工作的能力. 对软件漏洞进行分类是实现软件容错性的前提，它决定软件容错的方式. 软件漏洞是在硬件、软件、协议的具体实现或系统安全策略上(主要是人为)存在的缺陷，从而可以使攻击者能够在未授权的情况下访问或破坏系统. 目前，很多大型安全机构分别对软件漏洞进行分类，有 Microsoft 公司①、CVE 安全组织②和 Fortify Software 公司[17]等，本文采用 Fortify Software 的分类策略，对软件漏洞的分类以代码缺陷为基础，通过对代码缺陷的分类，间接地实现了对软件漏洞的分类. 同类代码缺陷引起的软件漏洞为一类，并在两个层次上进行归纳：(1) 在程序编码层次上归纳出软件的实现过程中的各种代码缺陷；(2) 根据漏洞攻击特征，把具有相同特征的代码缺陷归为一类. 由此，把代码缺陷分为 8 类，根据重要性高低依次排序为：(1) 输入有效性和表示；(2) API 滥用；(3) 安全功能；(4) 时间和状态；(5) 错误；(6) 代码质量；(7) 封装；(8) 环境.

如果对一个软件漏洞发起攻击或该漏洞在特定运行环境下出现异常，将可能会引起软件网络的大规模级联效应. 复杂网络的级联动力学研究表明[18]，故障传播特性、系统抗毁性与网络拓扑结构密切相关，即：无标度网络对随机故障具有很好的鲁棒性，但在基于度的蓄意攻击下存在脆弱性. 但是，目前从复杂网络角度评估软件容错性的研究还很少，因此有必要揭示复杂软件网络的级联故障传播规律，有助于软件的开发和部署，运行过程中因势利导，使漏洞得到有效控制和防护.

3　系统模型

大量研究人员集中在探讨软件系统的结构特性[7,10-11,19]、形成机理[2,20]、演化模型[4,21]和软件复杂性度量[10,22-24]，本文在对软件进行网络拓扑建模的基础上，进一步构建级联故障模型，探讨故障在软件网络中的传播情况.

3.1　网络拓扑模型

将软件网络表示为具有 n 个节点，e 条边的加权有向图 G，记为 $G=(V,E)$，其中 V 是节点集合，V 中的每个元素 v_i 代表软件源代码中的一个函数，E 是边集合，E 中的每个元素 $\langle v_i,v_j\rangle$ 是一个有序对，当且仅当 v_i 调用 v_j 时，$\langle v_i,v_j\rangle\in E$，即 $v_i\rightarrow v_j$.

在现有的大多数基于函数实体的软件网络模型中，节点间只有“相连”和“不相连”两种连接方式对应着函数间的“调用”和“不调用”两种关系，但这并

① Microsoft Corporation. Microsoft Security Response Center Security Bulletin Severity Rating System. http://www.microsoft.com/technet/security/bulletin/rating.mspx. Nov, 2002

② Christey S, Martin R A. Vulnerability Type Distributions in CVE. http://www.cve.mitre.org/docs/vuln-trends/index.html. May, 2007

不能准确地反映软件函数间的调用关系,实际上不同函数间连接的紧密程度也是不同的.因此,为每条有向边设置一个权重 w_{ij},用以表征函数间的调用频度,$w_{ij}\in[1,\infty)$.同时,定义一个 $n\times n$ 的邻接矩阵 $[a_{ij}]$,如果在源代码中,函数 v_i 调用函数 v_j 共 w_{ij} 次则 $a_{ij}=w_{ij}$,否则 $a_{ij}=0$.节点集合 V 不包括库函数.

3.2 级联故障模型

漏洞被触发后称之为故障,故障发生时对软件系统正常运行的影响程度称之为故障强度,实验中采用 Fortify Software 公司[17]对代码漏洞的分类方式定义故障强度,记为 λ,$\lambda\in[1,8]$,等级越高,意味着故障影响到其它函数的可能性越大.

软件网络中每个节点(函数)对故障的处理能力称为节点容错能力,记为 ρ.实际测试过程中采用的函数质量评估方法应与漏洞分类方法相一致,因此参考故障强度,容错能力也被划分为 8 个等级,即 $\rho\in[1,8]$,等级越高表示节点容错能力越强,越不易被软件故障所影响.本文设定两种容错能力分配方式:随机分配,各函数的容错能力符合均值为 ρ 的泊松分布;优先分配,各函数的容错能力与它拥有的调用关系数目(度)和调用关系紧密程度之和(强度)成正比例关系,即

$$\rho_i=\left\lceil\frac{k_i}{k_{\max}}\times\frac{s_i}{s_{\max}}\times 8\right\rceil \tag{1}$$

$$s_i=\sum_j^n a_{ij} \tag{2}$$

其中,k_i 和 s_i 分别表示节点 i 的度和强度(权重和),$k_{\max}$ 和 $s_{\max}$ 分别表示网络中最大度和最大强度.

触发漏洞有两种方式:随机触发,在图 G 中随机地选取 q 个节点作为初始故障节点,例如,由于运行环境的偶然变化导致函数功能异常;恶意触发,从图 G 中选择入度最大的前 q 个节点作为初始故障节点,例如,黑客发起的攻击.

在软件系统运行的过程中,如果函数 i 出现故障,例如内存溢出,那么该故障可能通过函数间的调用或依赖关系以某一概率传播至函数 j,而后者容错能力的强弱决定着该故障能否导致函数 j 失效,继而影响着调用和依赖函数 j 的其它函数.不失一般性,假定软件运行时一个或多个函数会同时发生强度相同的初始故障,当容错能力为 ρ_i 的函数 i 调用已发生强度为 λ_j 的故障的函数 j 时,本文规定当 $\lambda_j\geq\rho_i$ 时,函数 i 是否会受到该故障的影响仅依赖于函数 i 和函数 j 之间调用关系的紧密程度;当 $\lambda_j<\rho_i$ 时,故障能否感染函数 i 则依赖于两个函数的紧密程度和故障强度与容错能力之比的共同作用.因此,节点 i 的故障感染概率 P_i 表示为

$$P_i=W_{ij}\times\beta_{ij} \tag{3}$$

其中,

$$W_{ij}=\frac{w_{ij}}{w_{\max}} \tag{4}$$

$$\beta_{ij}=\begin{cases}1, & \lambda_j\geq\rho_i\\ \lambda_j/\rho_i, & \lambda_j<\rho_i\end{cases} \tag{5}$$

$w_{\max}$ 为图 G 的最大权重,即最大的调用次数,W_{ij} 表示函数 i 与函数 j 的调用关系在软件网络中发生的概率,即调用关系的紧密程度,这样设计的原因是因为函数被调用的次数越多意味着它具有的漏洞越易被触发,$W_{ij}\in(0,1]$.

发生故障的函数只有当被其它函数调用时才有可能将故障传播出去,因此,故障的一次传播规则定义如下:遍历所有已发生故障的节点,如果指向故障节点有向边的起始节点是非故障节点,则根据式(3)计算感染概率,并以此概率感染该节点,直至所有非故障邻居节点计算完毕.

4 实验分析

本文实验选取了 3 款大小和用途各异的开源软件 SNLC[1]、Sockets① 和 NotePad++②.Sockets 是 Berkeley 大学提供的一个套接字库,NotePad++是一款出色的代码编辑器.表 1 列出了 SNLC、Sockets 和 NotePad++的函数网络拓扑统计信息.从表 1 中可以看出,SNLC 中各节点之间的联系(2.28)较 Sockets(1.68)和 NotePad++(1.97)更为紧密.为便于理解,附录 A、B 和 C 中分别给出了规模最小的 SNLC 的原始函数调用关系网络、处理后的拓扑网络和节点编号与函数名称的对应关系.从附录 B 所示的拓扑图中可以发现,整个软件网络被 88 号函数分割为两个大的社团,而 88 号函数正好是 main 函数.

表 1 SNLC、Sockets、NotePad++节点和边的统计信息

软件名称	节点(函数)数	边(调用关系)数	平均出(入)度
SNLC	297	678	2.28
Sockets	710	1192	1.68
NotePad++	3117	6144	1.97

表 2 给出了 3 个软件网络弱连通图内的节点数和占总节点数的比例.从表中可以看出,Sockets 网络属于弱连通图的节点比例为 69%,这是 Sockets 软件性质决定的.Sockets 是一个套接字库,多数函数实现独立功能,不需要依赖于其它函数,同时,Sockets 提供了很多测试函数,也是"独立"于其它函

① Sockets. http://sockets.sourceforge.net/, Nov. 2009
② NotePad++. http://notepad-plus-plus.org/, Jan, 2009

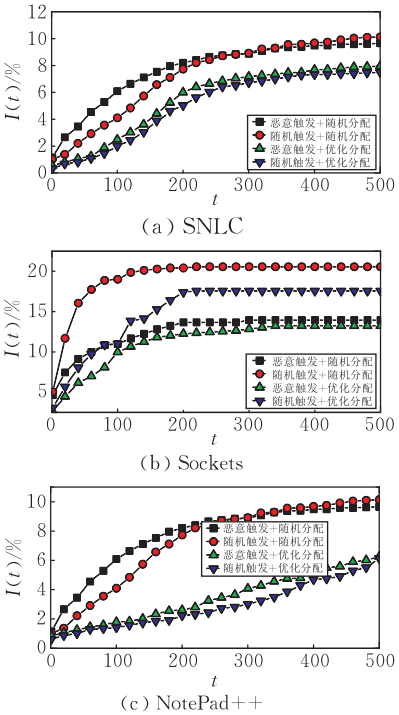
数的，因此不属于弱连通图. NotePad++实际网络属于弱连通图的节点比例为 73%，分析其源代码发现，NotePad++借助了一个插件 scintilla，它提供的函数仅有一部分被 NotePad++使用，但是网络建模时包括了 scintilla 的所有函数，因此弱连通图节点比例为 73%. 初始故障节点从弱连通图中选取，每次实验的循环次数为 500，每次模拟的最终结果对应着 50 次实验的平均值.

表 2　3 个软件网络的弱连通图统计

软件名称	弱连通图节点数	所占比例/%
SNLC	291	98
Sockets	493	69
NotePad++	2283	73

4.1　故障触发方式与容错能力分配方式

图 2 给出了故障触发方式和容错能力分配方式对级联故障在 3 种软件中传播的影响. 使用 $I(t)$ 表示 t 时刻发生故障的节点比例. 可以看出，恶意触发的故障传播速度略大于随机触发的故障传播速度. 这是因为恶意触发选择的初始故障节点是入度大的节点，这些节点与更多的节点相连，故障更易快速传播；而在随机触发下的初始故障节点是随机选择的，这些节点的入度平均分布，因此在“一跳”内，故障传播的范围小于恶意触发，导致传播速度没有恶意触发快. 同时可以发现，容错能力优先分配的软件网络比随机分配的软件网络具有更强的容错性，这表明



(a) SNLC

(b) Sockets

(c) NotePad++

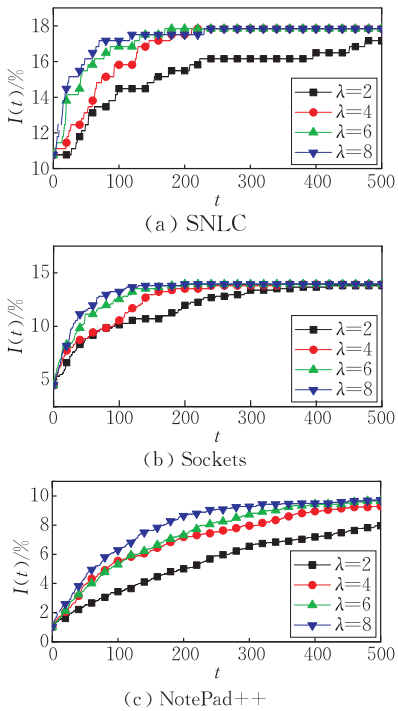
图 2　故障触发方式和容错能力分配方式对级联故障的影响($q=32$，$\lambda=8$，$\rho=4$)

那些具有高节点度(调用关系)和高强度(紧密程度)的函数对整个系统稳定性贡献更大. 这里说明一点，恶意攻击选择的初始目标仅是入度大的，而优先分配法则同时考虑节点度和强度，因此，恶意攻击选择的节点不一定是容错能力强的节点.

软件系统在故障被恶意触发、容错能力被随机分配的情况下，表现出最差的稳定性. 因此接下来考察在该情况下各模型参数对系统稳定性的影响.

4.2　故障强度

图 3 给出了故障强度对级联故障在 SNLC、Sockets 和 NotePad++ 3 种网络结构中传播的影响. 3 种软件对故障强度表现出一致的反应，即故障强度的增加会促使级联故障传播速度加快，对于规模大的软件而言(例如：NotePad++)，也会导致最终发生故障的节点数目也相应地增加. 但在各种故障强度下，即便是最高的 8 级，最终的感染规模也不超过 20%.



(a) SNLC

(b) Sockets

(c) NotePad++

图 3　故障强度对级联故障的影响($q=32$，$\rho=4$)

4.3　初始故障数目

图 4 给出了初始故障数目对级联故障在 SNLC、Sockets 和 NotePad++的 3 种网络结构中传播的影响. 少数的初始故障节点并不会诱发级联故障的大规模爆发，这也说明软件内部是允许存在少量错误的，他们的触发并不会对软件的正常工作带来巨大的影响，但当初始故障节点数目增加时，级联故障会逐步蔓延开来，但最终被感染的节点比例依然不会超过 20%.

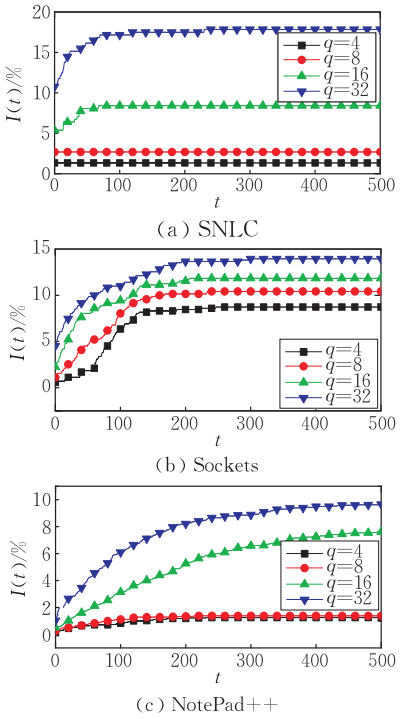


图 4 初始故障数目对级联故障的影响($\lambda=8,\rho=4$)

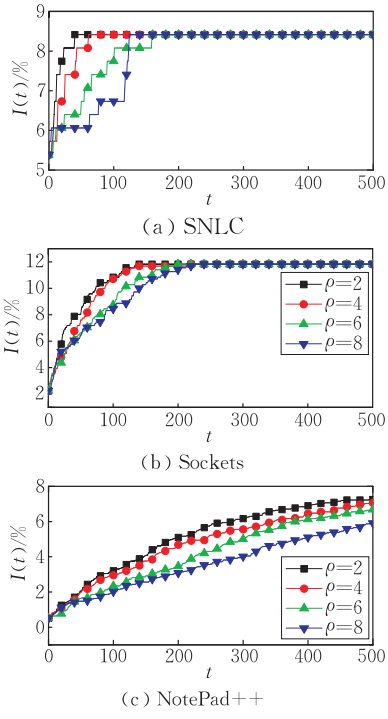


图 5 容错能力对级联故障的影响($q=16,\lambda=5$)

4.4 容错能力

图 5 展示了容错能力对级联故障在 SNLC、Sockets 和 NotePad++的 3 种网络结构中传播的影响. 级联故障在 3 种软件网络中对于提升的容错能力均表现出了相似的规律,即节点平均容错能力越大,故障传播速度越慢,稳态时感染节点的个数越少. 这也说明加强函数异常处理功能将提升软件整体质量和容错性,即便发生异常情况时,也不会影响大多数其它函数或模块的正常运行.

4.5 故障传播快照

图 6~8 分别给出各软件网络在故障被恶意触发、容错能力被随机分配情况下,当 $t=0,50,100$ 和 200 时的级联故障快照,便于更加直观地观察级联故障的传播过程,隐藏了节点间的连接关系. 从中可以看出,一旦漏洞被触发,发生软件故障后,该故障会随着函数间的调用关系逐渐扩散,最终演变为级联故障. 对比各软件图中故障节点(浅色)增加的比例可发现,级联故障在 3 种软件网络中的前 50 次迭代传播最为迅速.

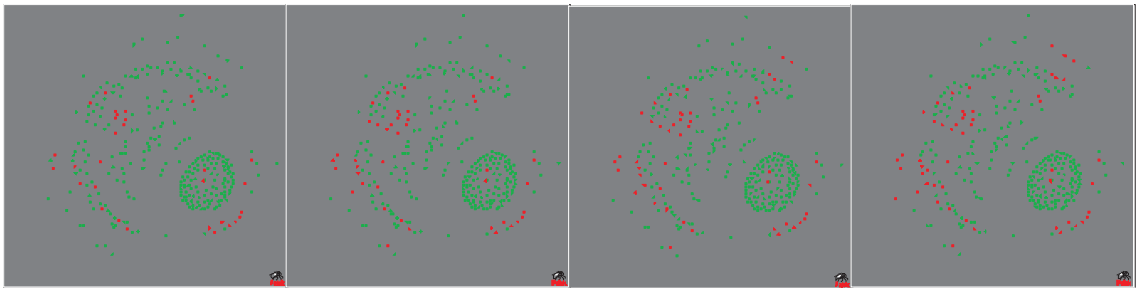


图 6 SNLC 不同时刻的级联故障传播快照($q=32,\lambda=8,\rho=4$. 从左至右依次为 $t=0,50,100$ 和 200)

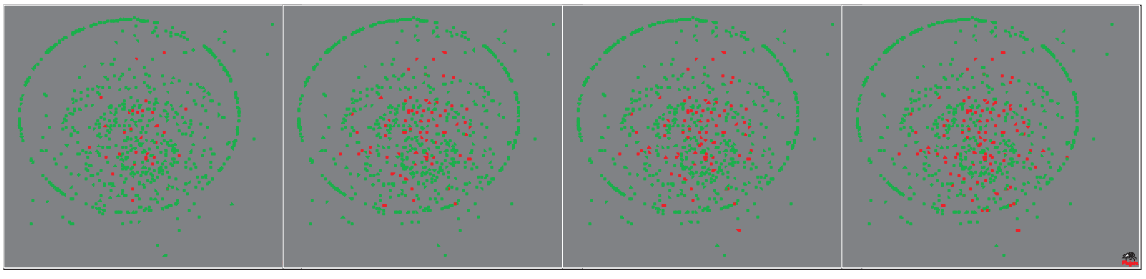


图 7 Sockets 不同时刻的级联故障传播快照($q=32,\lambda=8,\rho=4$. 从左至右依次为 $t=0,50,100$ 和 200).

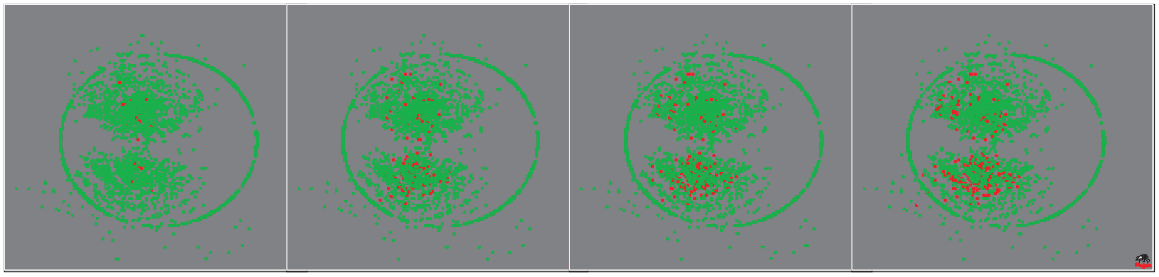


图 8　NotePad++不同时刻的级联故障传播快照($q=32,\lambda=8,\rho=4$.从左至右依次为 $t=0,50,100$ 和 200)

4.6　节点最终未被感染的原因分析

实验过程中发现,有一些节点虽属于连通图,但直到级联故障传播稳定后,依然没有被感染.本文通过对规模最小的 SNLC 在 $q=16,\lambda=5,\rho=4$ 随机触发下的故障传播数据进行跟踪分析后,发现了如图 9 所示的 4 种具有代表性的宏观原因.更细致的微观原因可以试图从这 4 种宏观原因入手分析,将在后续研究工作中给出结果.

(1)节点自身容错能力强,如图 9(a)所示.例如,编号为 38 的节点,容错能力为 7,具有较强的容错能力,而故障强度为 5,所以不易受故障传播的影响;

(2)周围节点的容错能力强,如图 9(b)所示.节点 65 的容错能力为 4,它的两个邻居节点 79 和 100 的容错能力分别为 6 和 5,具有较强的容错能力,所以故障传播到节点 65 处为止,使调用 79 和 100 节点的其它容错能力弱的节点也不会被感染;

(3)没有入度的节点,如图 9(c)所示.节点 176 将故障传播给节点 216,节点 216 的容错能力为 2,本应该继续传播故障,但该节点没有入度,即没有其它函数调用 216 号函数,所以级联故障传播到 216 号处为止;

(4)有限的迭代次数,这种情况在 NotePad++这种规模较大的软件中更为明显.观察图 9(d),当故障传播到节点 5 后,它的容错能力为 3,但模拟时间已到,故障不再继续传播.

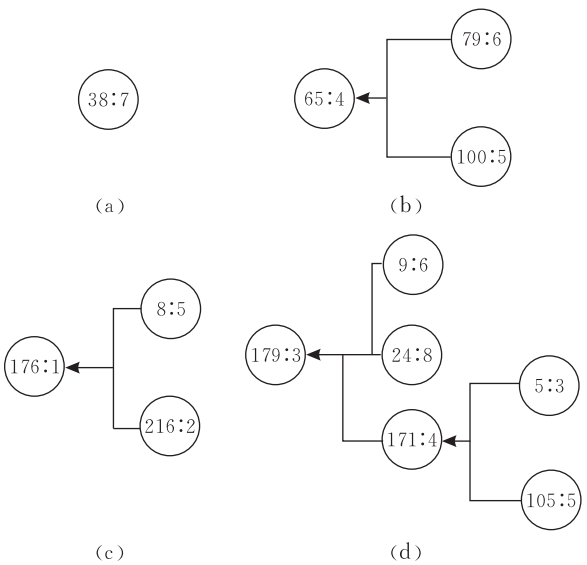


图 9　节点未被感染的原因示例(圆圈代表节点,里边的数字为"节点号:容错能力",对比附录 A、B 和 C 可以找到各节点的编号、函数名和连接关系)

图 9 节点未被感染的原因示例.圆圈代表节点,里边的数字为"节点号:容错能力",对比附录 A、B 和 C 可以找到各节点的编号、函数名和连接关系.

5　结束语

本文基于软件函数间的调用关系,引入故障强度和容错能力对复杂软件的级联故障进行建模,模拟软件运行时的故障传播行为.通过对 3 个实际软件网络的大量模拟实验,揭示了触发方式、容错能力分配方式、故障强度、初始故障数目和容错能力大小等因素对级联故障传播的影响,并分析了造成这些影响的原因,也探讨了节点未被最终感染的可能原因.

模拟结果揭示出级联故障的传播速度和范围同故障强度、初始故障数目、容错能力密切相关,即:弱的故障强度、少的初始故障数目和高的容错能力均会减缓级联故障的传播速度,缩小波及范围,提高软件质量.

对复杂软件的级联故障进行建模有助于增进对软件复杂性和容错性的认识,为提高软件质量提供帮助.在本文的级联故障模型中,故障强度在整个传播过程中始终保持不变,但由于各函数容错能力的作用和差异,该故障的强度应随着函数的调用而产生变化,例如,如果某函数没有异常处理环节,那么该故障传播至此后强度应该被增加,反之则减弱.下一步工作将设计故障强度的更新规则,基于本文的研究结果,并结合具体的软件工程开发与测试技术,开发复杂软件容错性的检测工具.

参 考 文 献

[1] Wang Jian, Liu Yan-Heng, Mei Fang, Zhang Cheng. Modeling cascading failures for Internet based on congestion effects. Journal of Computer Research and Development, 2010, 47(5): 772-779(in Chinese)
(王健, 刘衍珩, 梅芳, 张程. 基于网络拥塞的 Internet 级联故障建模, 计算机研究与发展, 2010, 47(5): 772-779)

[2] Valverde S, Cancho R F, Solé R V. Scale free networks from optimal design. Europhysics Letters, 2002, 60(4): 512-517

[3] Lv Jin-Hu, Wang Hong-Chun, He Ke-Qing. Complex dynamical networks and their applications in software engineering. Journal of Computer Research and Development, 2008, 45(12): 2052-2059(in Chinese)
(吕金虎, 王红春, 何克清. 复杂动力网络及其在软件工程中的应用. 计算机研究与发展, 2008, 45(12): 2052-2059)

[4] He Ke-Qing, Peng Rong, Liu Jing et al. Design methodology of networked software evolution growth based on software pattern. Journal of Systems Science & Complexity, 2006, 19(2): 157-181

[5] Li De-Yi, Liu Kun, Sun Yan, Han Ming-Chang. Emergent computation: Virtual reality from disordered clapping to ordered clapping. Science in China Series F: Information Sciences, 2008, 51(5): 449—459(in Chinese)
(李德毅, 刘坤, 孙岩, 韩明畅. 涌现计算:从无序掌声到有序掌声的虚拟现实. 中国科学: E 辑, 2007, 37(10): 1248-1257)

[6] Cai Kaiyuan, Yin Beibei. Software execution processes as an evolving complex network. Information Science, 2009, 179(12): 1903-1928

[7] Myers C R. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. Physical Review E, 2003, 68(4): 046116

[8] McCabe T J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2(4): 308-320

[9] Helstead M H. Elements of Software Science. Amsterdam: Elsevrer Press, 1977

[10] Alessandro P S de Moura, Lai Y-C, Adilson E M. Signatures of small-world and scale-free properties in large computer programs. Physical Review E, 2003, 68(1): 017102

[11] Zheng X, Zeng D, Li H et al. Analyzing open-source software systems as complex networks. Physica A, 2008, 387(24): 6190-6200

[12] Li D, Han Y, Hu J. Complex network thinking in software engineering//Proceedings of the International Conference on Computer Science and Software Engineering. Wuhan, China, 2008: 264-268.

[13] Zhu M, Zhang H, Qi W et al. The measurement and evaluation for large-scale object-oriented software system//Proceedings of the International Conference on Hybrid Intelligent Systems. Shenyang, China, 2009: 70-73

[14] Ma Y, He K, Du D et al. A complexity metrics set for large-scale object-oriented software systems//Proceedings of the IEEE International Conference on Computer and Information Technology. Seoul, Korea, 2006: 189

[15] Yao Y, Huang S, Hong Y et al. Structural characteristic analysis of large scale object-oriented software and its evolution based on complex network theory//Proceedings of the International Conference on Information Engineering. Taiyuan, China, 2009: 321-324

[16] Sun S, Xia C, Chen Z et al. On structural properties of large-scale software systems from the perspective of complex networks//Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery. Tianjin, China, 2009: 309-313

[17] Tsipenyuk K, Chess B, McGraw G. Seven pernicious kingdoms: A taxonomy of software security errors. IEEE Security & Privacy, 2006, 3(6): 81-84

[18] Wang Jian, Liu Yan-Heng, Zhang Cheng, Li Cheng-Yue. Analyzing and modeling cascading dynamics of Internet. Journal of Software, 2010, 21(8): 2050-2058(in Chinese)
(王健, 刘衍珩, 张程, 李成岳. Internet 级联动力学分析与建模, 软件学报, 2010, 21(8): 2050-2058)

[19] Wheeldon R, Counsell S. Power law distributions in class relationships//Proceedings of the International Workshop on Source Code Analysis and Manipulation. Amsterdam, The Netherlands, 2003: 45-54

[20] Solé R V, Valverde S, Sole R V et al. Information theory of complex networks: On evolution and architectural constraints//Eli Ben-Naim, Hans Frauenfelder, Zoltan Toroczkai. Lecture Notes in Physics Special volume on Complex Networks, 2004, 650: 189-207

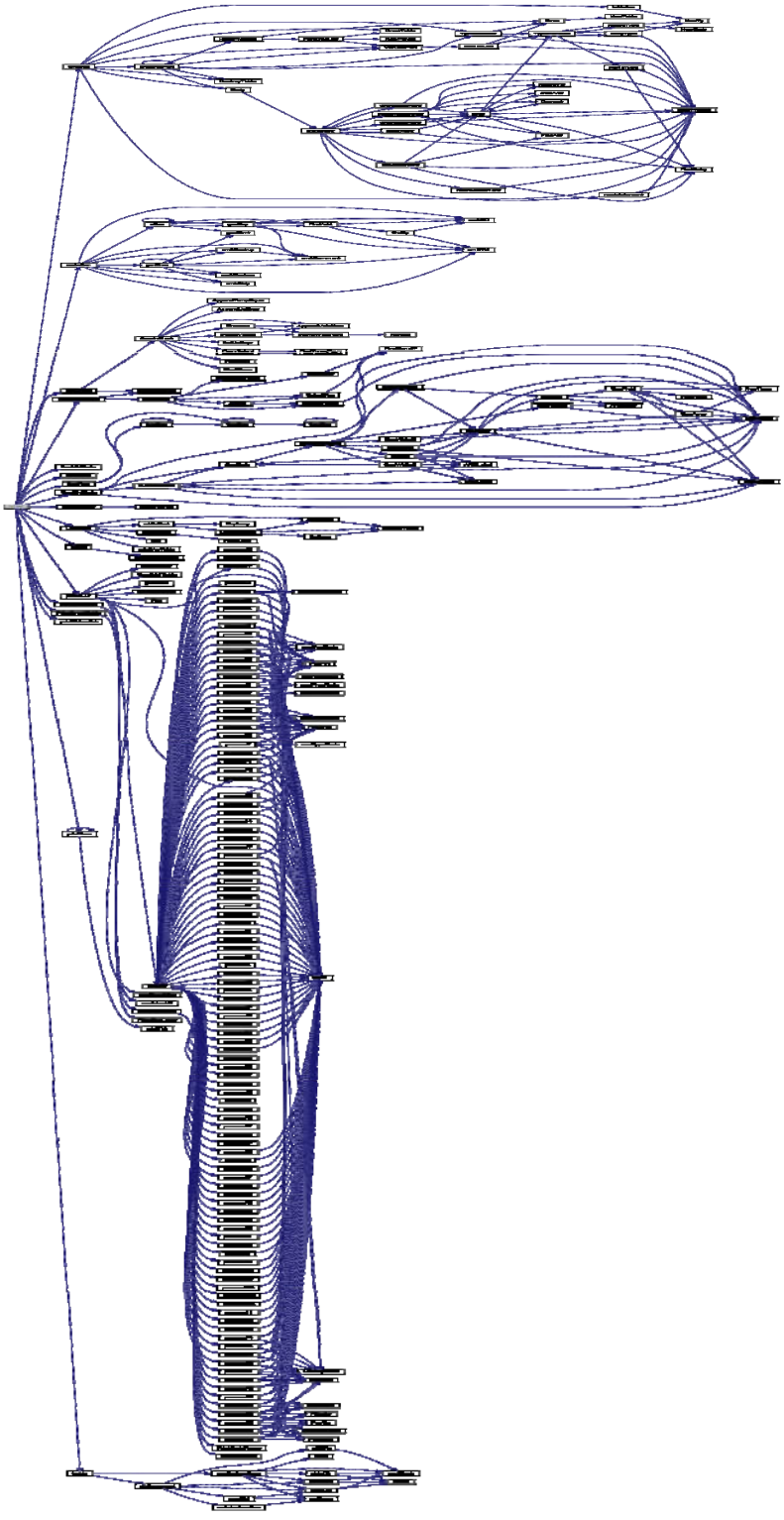
[21] Valverde S, Solé R V. Logarithmic growth dynamics in software networks. Europhysical Letter, 2005, 72(5): 858-864

[22] Vasa R, Schneider J G, Woodward C et al. Detecting structural changes in object oriented software systems//Proceedings of the International Symposium on Empirical Software Engineering. Noosa Heads, Australia, 2005: 479-486

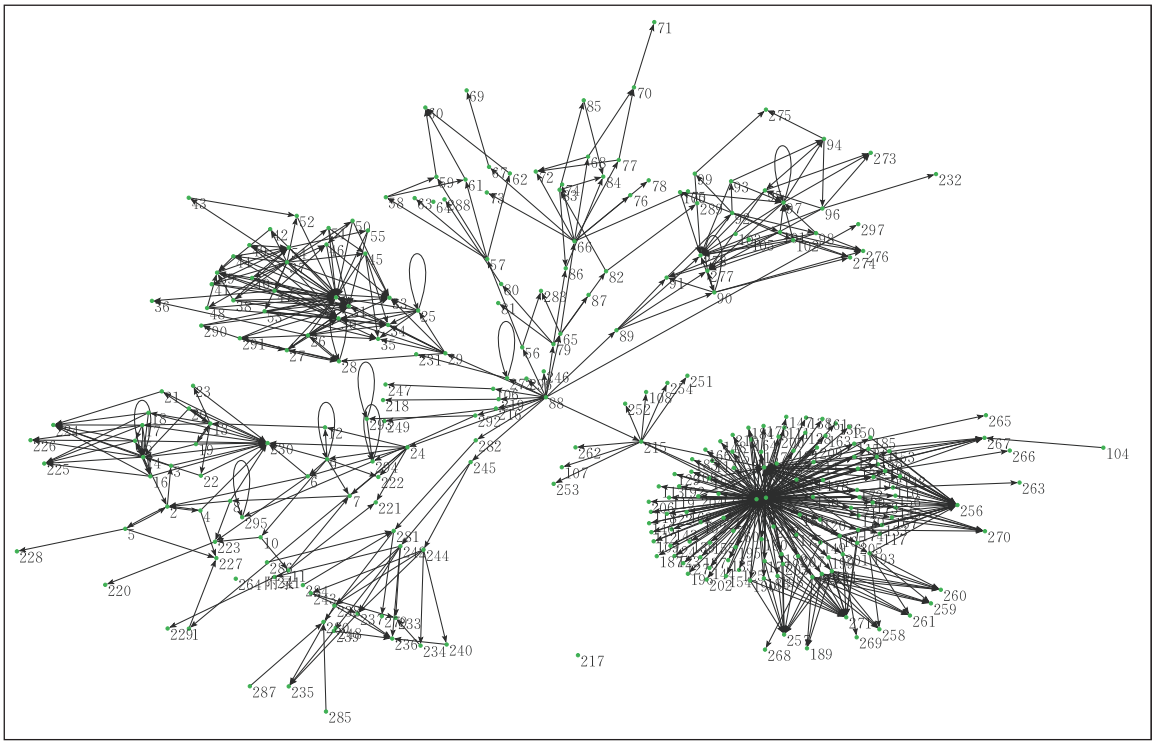
[23] Liu J, He K Q, Ma Y T et al. Scale flee in software metrics//Proceedings of the International Computer Software and Applications Conference. Chicago, Illinois, 2006: 229-235

[24] Li Bing, Wang Hao, Li Zeng-Yang, He Ke-Qing, Yu Dun-Hui. Software complexity metrics based on complex networks. Acta Electronica Sinica, 2006, 34(12A): 2371-2375 (in Chinese)
(李兵, 王浩, 李增扬, 何克清, 余敦辉. 基于复杂网络的软件复杂性度量研究. 电子学报, 2006, 34(12A): 2371-2375)

附录 A.　SNLC 函数调用关系图.



附录 B. 处理后的 SNLC 节点拓扑图.



附录 C. SNLC 节点编号与函数名的对应关系.

1 initialize	28 cGen	55 FindSp	82 whileEnd	109 process1	136 process28
2 TypeProcess	29 codeGen	56 ConstOptimize	83 LoopOutside	110 process2	137 process29
3 nameType	30 emitComment	57 OptiBlock	84 SearchTable	111 process3	138 process30
4 arrayType	31 emitRO	58 ArithC	85 DelItem	112 process4	139 process31
5 recordType	32 emitRM	59 SubstiArg	86 AddTable	113 process5	140 process32
6 TypeDecPart	33 emitSkip	60 FindConstT	87 printVarTable	114 process6	141 process33
7 VarDecPart	34 emitBackup	61 AppendTable	88 main	115 process7	142 process34
8 varDecList	35 emitRestore	62 DelConst	89 GenMidCode	116 process8	143 process35
9 procDecPart	36 emitRM_Abs	63 printConstTable	90 GenProcDec	117 process9	144 process36
10 HeadProcess	37 codeGen	64 NewVN	91 GenBody	118 process10	145 process37
11 ParaDecList	38 arithGen	65 ECCsave	92 GenStatement	119 process11	146 process38
12 Body	39 operandGen	66 SaveInBlock	93 GenAssignS	120 process12	147 process39
13 statement	40 aaddGen	67 EquaSubsti	94 GenVar	121 process13	148 process40
14 Expr	41 readGen	68 Process	95 GenArray	122 process14	149 process41
15 arrayVar	42 writeGen	69 FindTempEqua	96 GenField	123 process15	150 process42
16 recordVar	43 returnGen	70 SearchValuNum	97 GenExpr	124 process16	151 process43
17 assignstatement	44 assigGen	71 IsEqual	98 GenCallS	125 process17	152 process44
18 callstatement	45 labelGen	72 AppendValuNum	99 GenReadS	126 process18	153 process45
19 ifstatment	46 jumpGen	73 FindECC	100 GenWriteS	127 process19	154 process46
20 whilestatement	47 jump0Gen	74 AppendTempEqua	101 GenIfS	128 process20	155 process47
21 readstatement	48 valactGen	75 GenMirror	102 GenWhileS	129 process21	156 process48
22 writestatement	49 varactGen	76 AppendUsExpr	103 arrayType	130 process22	157 process49
23 returnstatement	50 callGen	77 SubstiVcode	104 varDecList	131 process23	158 process50
24 analyze	51 pentryGen	78 DelUsExpr	105 procDecPart	132 process24	159 process51
25 genProc	52 endprocGen	79 LoopOpti	106 parse	133 process25	160 process52
26 genStmt	53 mentryGen	80 whileEntry	107 CreatLL1Table	134 process26	161 process53
27 genExp	54 FindAddr	81 call	108 gettoken	135 process27	162 process54

163 process55
164 process56
165 process57
166 process58
167 process59
168 process60
169 process61
170 process62
171 process63
172 process64
173 process65
174 process66
175 process67
176 process68
177 process69
178 process70
179 process71
180 process72
181 process73
182 process74
183 process75
184 process76
185 process77
186 process78
187 process79
188 process80
189 Priosity
190 process81
191 process82
192 process83
193 process84
194 process85
195 process86
196 process87
197 process88
198 process89
199 process90
200 process91
201 process92
202 process93
203 process94
204 process95
205 process96
206 process97
207 process98
208 process99
209 process100
210 process101
211 process102
212 process103
213 process104
214 predict
215 parseLL1
216 getTokenlist
217 PrintFieldChain
218 PrintOneLayer
219 PrintSymbTable
220 NewTable
221 CreatTable
222 DestroyTable
223 Enter
224 FindEntry
225 FindAttr
226 Compat
227 NewTy
228 NewBody
229 NewParam
230 ErrorPrompt
231 printTab
232 FindField
233 opClass
234 writeInstruction
235 getCh
236 nonBlank
237 getNum
238 getWord
239 skipCh
240 atEOL
241 error
242 readInstructions
243 stepTM
244 doCommand
245 tmain
246 printTokenlist
247 ReadNextToken
248 copyString
249 ChainToFile
250 Push
251 Pop
252 readStackflag
253 readstackN
254 readstackT
255 PushPA
256 PopPA
257 PushOp
258 PopOp
259 ReadOpStack
260 PushNum
261 PopNum
262 newRootNode
263 newPheadNode
264 newDecANode
265 newTypeNode
266 newVarNode
267 newDecNode
268 newProcNode
269 newStmlNode
270 newStmtNode
271 newExpNode
272 printTree
273 NewTemp
274 NewLabel
275 ARGAddr
276 ARGLabel
277 ARGValue
278 GenCode
279 PrintCodeName
280 PrintContent
281 PrintOneCode
282 PrintMidCode
283 DivBaseBlock
284 PrintBaseBlock
285 printValuNum
286 printUsbleExpr
287 printTempEqua
288 PushLoop
289 PopLoop
290 findSp
291 FindAdd
292 freeTree
293 freeDec
294 freeStm
295 freeExp
296 freeTable
297 freeMidCode

WANG Jian, born in 1981, Ph. D. candidate, lecturer. His main research interests include complex networks and network security.

LIU Yan-Heng, born in 1958, Ph. D., professor, Ph. D. supervisor. His main research interests include mobile IP and QoS.

LIU Xue-Lian, born in 1984, M. S. candidate. Her main research interests focus on trusted software.

Background

As computers are widely utilized in various systems including civil and safety-critical applications, software faults have become the major factor that causes critical problems. Hence, there exists an increasing demand of evaluating the robustness. Cascading fault models can provide quantitative measures of the reliability of software systems during software running processes.

This research is supported by National Nature Science Foundation under Grant 60973136 and 61073164, the China-British Columbia Innovation and Commercialization Strategic Development under Grant 2008DFA12140, European BRIDGING THE GAP Erasmus Mundus Project under Grant 155776-EM-1-2009-1-IT-ERAMUNDUS-ECW-L12, the Jilin University 985 Graduate Student Innovation Foundation under Grant 20101029, and the Fundamental Research Funds for Jilin University under Grant 201103136.