第41卷 第6A期
2014年6月

计 算 机 科 学
Computer Science

Vol. 41 No. 6A
June 2014

可信编译器构造的翻译确认方法简述

刘 洋[1] 杨 斐[1] 石 刚[1,2] 闫 鑫[1,3] 王生原[1] 董 渊[1]

(清华大学计算机科学与技术系 北京 100084)[1] (新疆大学信息科学与工程学院 乌鲁木齐 830046)[2]

(太原理工大学计算机学院 太原 030021)[3]

摘 要 编译器的安全可信问题日益引起重视,特别是在安全关键系统中,编译器的误编译将会造成重大的损失。消除误编译的传统方法是大量的测试,但是测试难以达到完全覆盖,并不能充分地保证编译器的安全可信。近年来,形式化验证方法被成功用于可信编译器的构造中。一种方法是对编译器本身进行形式化验证,经过严密的证明,可杜绝误编译的发生。然而,这种方法可能"冻结"编译器的设计,阻碍编译器未来可能的优化和完善。翻译确认是另外一种用于可信编译器构造的形式化方法,它避免了对编译器自身的验证,有很好的可重用性,近年来在编译器验证领域得到了广泛研究,已取得令人瞩目的成果。介绍了翻译确认方法的概念及研究进展。

关键词 可信编译器,形式化验证方法,翻译确认,确认器

中图法分类号 TP314 **文献标识码** A

Brief Overview of Translation Validation Method in Trusted Compiler Construction

LIU Yang[1] YANG Fei[1] SHI Gang[1,2] YAN Xin[1,3] WANG Sheng-yuan[1] DONG Yuan[1]

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[1]

(School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China)[2]

(School of Computer Science, Taiyuan University of Technology, Taiyuan 030021, China)[3]

Abstract There is a growing awareness of the security and reliability of compilers in recent years. Especially in the safety-critical systems, the miscompilation can make a huge damage. The conventional approach to weeding out miscompilation is heavy test, but test is impossible to achieve full cover and can not guarantee that compiler is safe and trustworthy. In recent years, formal verification method has been successfully used to construct the trusted compiler. One approach is to perform formal verification for compiler itself. It can put an end to miscompilation by strict proof. However, this approach tends to "freeze" the compiler design, and discourages any future improvements and revisions. Translation validation is another formal method that can be used to construct trusted compiler, which avoids the verification for the compiler itself, and has good reusability. It has been widely researched in the compiler verification field, and has already obtained remarkable achievements. The concept and research progress of translation validation are discussed in this paper.

Keywords Trusted compiler, Formal verification method, Translation validation, Validator

1 引言

在一些安全关键领域中,例如航天、核工业等,编译器的安全可信有着至关重要的作用,编译错误可能会带来灾难性的后果。然而随着计算机技术的发展,对编译的要求越来越高,例如高级语言编译器中增加了大量的优化,使得编译器变得越来越复杂,这对如何构造可信编译器形成了更大的挑战。

可信编译器可通俗地理解为可信任的编译器,具体指标是指编译过程的正确性,编译器能确保其编译过程是正确无误的,不会带来误编译,目标代码能将源代码的特征正确、完整地实现,其语义与源代码保持一致。然而,有时因为编译器设计或实现的问题,编译器会产生误编译,导致目标代码与源代码行为或语义不一致,或者导致编译时崩溃。

构造可信编译器、消除误编译的传统方法有:1)测试,通过大量的测试来发现编译器的错误;2)通过编译器开发过程的严格质量控制来防范编译错误;3)人工检查生成的目标代码,查找目标代码与源代码行为是否一致。然而这3种方法都有着明显的缺陷:首先,由于编译器的复杂性,测试工作不容易展开,难以达到对编译器的全覆盖测试,且测试结果不仅受编译器的影响,还受源程序的影响,测试人员需花费大量的精力来区别是源程序本身的错误还是编译器的错误;其次,过程管理只能尽量避免在开发过程中人为引入的错误,而不能完全杜绝编译错误的发生;第三,人工检查代码不仅受到源程序规模的制约,而且现代的编译器中包含了大量的优化,其所

本文受国家自然基金项目(61170051,612702086,90818019)资助。

刘 洋(1983—),男,硕士生,主要研究方向为编译器形式化验证;杨 斐(1984—),男,硕士生,主要研究方向为编译器形式化验证;石 刚(1972—),男,博士生,主要研究方向为系统软件的形式化验证。

产生的目标代码与源代码之间的结构等都发生了变化，难以人工检查二者的一致性。

近年来，形式化验证方法被成功用于可信编译器的构造中。形式化验证方法从数学角度对编译器进行描述，对编译过程的语义和语言属性的等价性进行证明，能够充分地保证编译器可信。形式化验证方法在编译器验证中有多种实现形式，一种方法是对编译器本身进行形式化验证，经过严密的证明，可杜绝误编译的发生。然而这种方法有着一定的缺陷：它必须依据编译器的具体实现进行证明，因此对编译器的改变是非常敏感的，这就导致编译器的任何一个变化都将使原有的证明失效，以至于形式化验证"冻结"了编译器的设计，阻碍了编译器未来的优化和完善[3]。另外，此方法是直接对编译器的代码进行的，当编译器代码无法得到时，例如一些非开源的编译器，则无法采用此方法。

翻译确认是另外一种用于可信编译器构造的形式化方法，它通过证明源代码和目标代码的语义等价性来证明编译器的正确性，避免了对编译器自身的验证，同时又具有很好的可重用性，近年来在编译器验证领域得到了广泛研究，已取得令人瞩目的成果。

本文旨在介绍翻译确认方法及研究进展，第 2 节介绍翻译确认的提出及其主要概念；第 3 节介绍翻译确认的研究进展；第 4 节介绍翻译确认目前的主要研究方向；最后进行总结。

2 翻译确认方法的出现及其主要概念

1997 年，Cimatti[2] 提出了一个嵌入 VFRAME 组件的"嵌入式证明器"，用来证明 VFRAME 组件中的编译器正确性。每次编译器运行时，证明器接收当前的源程序和编译器生成的目标程序作为输入，通过分析比较二者的句法特性来证明编译器的正确性。证明器作为 VFRAME 组件的一部分，完全独立于编译器，证明器的设计不依赖于编译器，在编译时证明器自动运行，且不能影响编译器的性能。为了证明其有效性，"嵌入式证明器"自身必须被证明是正确的。

Cimatti 的"嵌入式证明器"具有形式化验证的特点，同时又不依赖于编译器，是一项开创性的工作。然而其又有一定的局限性，主要是证明器的证明是在句法层进行，所针对的源语言和目标语言要求具有很大的相似性，二者之间的转换非常简单直接，证明器所采用的证明技术比较简单。

1998 年，Pnueli[3] 第一次提出了翻译确认(translation validation)概念。翻译确认是一种用于确认编译器或代码生成器的源和目标之间的语义等价性的形式化方法，它通过证明源代码和目标代码的语义等价性来证明编译器的正确性。使用翻译确认方法需要构造一个确认器(validator)，确认器在编译器每一次运行后形式化地证明生成的目标代码是源代码的一个正确翻译。确认器不关心编译器的具体实现，只对编译器的源代码和目标代码进行处理，如果验证成功则编译继续进行，如果发现语义矛盾之处则输出一个警报或取消编译。编译器的形式化验证可以减弱为对确认器进行形式化验证工作(相关定理论述见文献[15])。相对于编译器而言，确认器的形式化验证工作是比较简单的，从而大大减轻了证明的难度及工作量。同时，由于确认器不关心编译器的具体实现，因此没有限制编译器的设计以及未来的优化完善等，且确认器是可重用的。

Pnueli 指出一个自动化的翻译确认器应该包括以下要素：(1)一个用于描述源语言和目标语言的公共语义框架；(2)基于公共语义框架形式化地建立的目标代码和源代码之间的"正确执行"定理；(3)一个有效的证明方法；(4)通过 analyzer 执行证明方法的自动化，如果成功则生成一个证明脚本；(5)一个证明检查器，用于对 Analyzer 产生的证明脚本进行检查。Pnueli 描述的翻译确认的概念图如图 1 所示[3]。

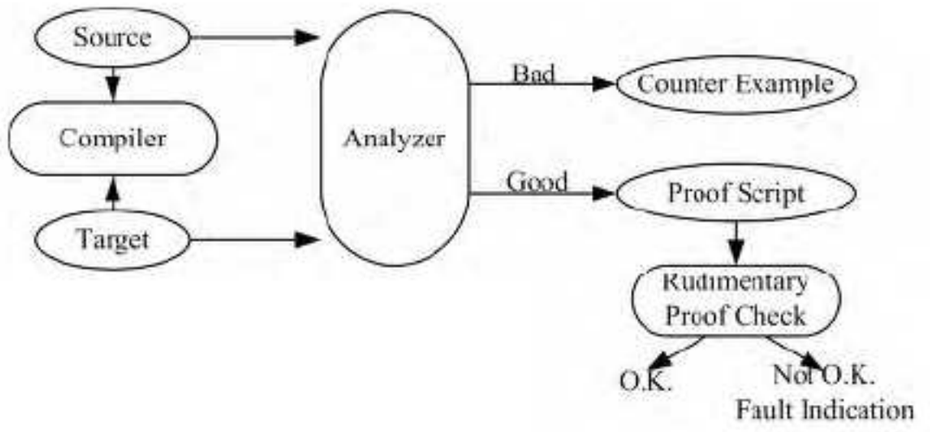

图 1 翻译确认概念图

Pnueli 对多时钟同步数据流语言 SIGNAL 到 C 的编译进行了翻译确认，采用了 STS(synchronous transition systems)同步转换系统作为公共语义框架，建立了源语言和目标 C 语言的语义模型，定义了二者之间的"求精"关系，从而建立了 SIGNAL 到 C 的语义等价性。在此基础上采用确认器对 SIGNAL 到 C 的编译过程进行翻译确认。

相比于 Cimatti 的"嵌入式证明器"，Pnueli 提出的翻译确认方法有一定的相似性，例如：都是以编译器的源代码和目标代码作为输入，都需要对证明器本身进行正确性证明，都是独立于编译器自动运行，但同时有着本质的区别：首先，翻译确认针对的是基于公共语义框架下源语言和目标语言的语义分析而不是句法层，并不要求源语言和目标语言之间具有相似性，例如 SIGNAL 到 C 就是同步数据流语言到非同步语言的转换，因而翻译确认能适用于更普遍的编译器；其次，翻译确认必须采用一定的确认算法来进行证明，针对不同的验证需求其算法也将千差万别，而不是简单的比较能完成的，例如文献[3]采用基于 STS 抽象计算模型上的求精。

3 翻译确认方法的研究进展

Pnueli 的工作说明翻译确认可以实现编译器的形式化验证，从而开辟了编译器验证的一条新思路。在 SIGNAL 到 C 的编译过程的翻译确认工作中，Pnueli 实现了非同构语言之间的翻译确认，然而这个编译过程仍旧是相对简单的，譬如其中并没有涉及到编译优化等。

随后，Pnueli 在原有工作的基础上继续实现了对两种同步语言到 C 的编译器的翻译确认[4]，这两个编译器中包含了大约 100 个优化规则，证明了翻译确认方法也适用于优化编译器，并实现了一个翻译确认器 CVT(code validation tool)。Zuck 在 Pnueli 的研究基础上将该确认器成功扩展应用于 TNI 和 inria 编译器[5]、包含有限优化集的编译器[7]以及具有广泛优化集的高级 EPIC 编译器[9-12]。

Pnueli 的翻译确认算法主要是依靠基于转换系统的证明条件生成以及定理证明，Necula 则另辟蹊径提出了一种基于抽象解释和静态分析的翻译确认算法。在现代的编译器中，编译器一般需要经过多次中间转换，例如 CompCert 编译器中采用了 9 种中间语言，经过了 14 个中间转换过程完成最终到目标代码的编译过程。Necula 不考虑整个编译器的编译，

而是将编译器的每一个编译过程分别考虑。文献[8]实现了一个用于 gcc2.7 编译器的编译确认基础设施(translation validation infrastructure,TVI),TVI 分别针对每个编译过程分别进行翻译确认,利用编译时提供的额外附加信息,采用 IL 中间语言分别描述编译前后的代码语义,提出了基于符号计算的两步推理算法和基于仿真关系判断程序等价性算法两种算法,可以可靠地确认 4 种 gcc 的过程内优化,包括:指令调度、公共子表达式消除、循环展开和寄存器分配,但仍然存在一定的误报率。Rival 使用符号转换函数来描述代码的行为,提出了一种针对 gcc3.0 无优化编译器的翻译确认方法[14]。

Pnueli 和 Zuck 的工作主要是针对同步数据流语言到 C 的翻译确认,Ryabtsev 在 Pnueli 工作的基础上实现了一种翻译确认工具 TVS(Translation Validation tool for Simulink),用以对非同步语言 Simulink 到 C 的代码生成器 Real-Time Workshop(RTW)编译过程进行翻译确认[13]。Ryabtsev 仍然采用 STS 计算模型建立 RTW 的源代码和目标代码语义模型,不同于 Pnueli 工作的是,Ryabtsev 并未建立二者的求精关系,而是由 TVS 确认器依据 RTW 建立源代码变量到目标代码变量之间的映射关系以及证明条件,然后借助于 SMT 定理证明器对 TVS 产生的证明条件进行证明。Ryabtsev 的工作主要集中于证明条件的建立,而具体的证明工作交由自动定理证明器完成,展现了一种不同于 Pnueli 的新思路,然而其确认器本身的正确性并未进行验证,正确性难以充分保证。

随着编译器的发展,现代的高级编译器中包含着大量的优化,翻译确认更多地应用于对局部编译优化的验证。在对编译优化的验证中,将确认器接收优化前和优化后的代码作为输入,建立证明条件,对二者的语义等价性进行证明,输出验证结果。

Barret[27]和 Fang[31]实现了一个针对 Intel 的 ORC 编译器和 EPIC 高级优化编译器的翻译确认器 TVOC,采用转换系统(Transition System,TS)描述优化前后的代码,依靠编译器产生的附加信息建立证明规则和验证条件,然后使用自动定理证明器 CVC Lite 进行证明,能有效地对结构保持的优化进行证明。但是 TVOC 不能用于非结构保持优化的证明。Goldberg 在 Barret 研究的基础上,将 TVOC 加以改进,为 TVOC 制定统一的证明规则,使之能适应非结构保持优化[32]。

Kanade[22]提出了一个原型证明系统(Prototype Verification System,PVS),定义了一套转换原语,用原语描述编译优化过程以及验证条件,在 PVS 框架下对编译优化进行正确性证明。

Tate 针对编译 Java 到 JVM 的 Soot 优化编译器提出了一种翻译确认算法:采用平等饱和过程构建输入代码和对应的优化代码求值图,如果经过饱和处理后两个图是相同的,则编译器的优化是正确的[17]。Stepp 在 Tate 工作的基础上,对算法进行改进,针对 LLVM 编译器的特性增加了相应的组件,将该算法成功应用于 LLVM 编译器[34]。

Tristan 实现了一种"标准求值图确认器"(LLVM-MD)用于对 LLVM 编译器的过程内优化进行翻译确认,采用 SSA 形式分别描述优化前后的代码,构造求值图,通过变量的定义分别计算求值图,然后应用标准化规则和最大化共享简化求值图直至使得两个求值图的值合并成一个单节点(验证通过),或者已经不能进行任何标准化为止(验证未通过)[18]。

编译器优化分为过程内优化以及过程间优化,随着对翻译确认方法研究及应用的深入,翻译确认已能适用于大部分过程内优化,然而对于过程间优化仍不可行。Pnueli 和 Zask 尝试用翻译确认对编译器的过程间优化进行验证,提出了一种基于转换图(transition graphs)演绎的过程间翻译确认算法,其能应用于 GCC、ORC 以及 LLVM 编译器[33],但误报率和性能仍存在一定问题。

翻译确认方法确认的准确率或者说误报率一直是一个非常值得关注的问题,在目前的应用中,翻译确认方法不仅应用于对编译器的验证,同时还可以用于编译器的测试等工作,但它并不能完全取代其他的验证或测试工作,很大一部分原因就是存在误报率,虽然很小,但也不能忽略。

Trsitan 做出了杰出的工作,他将编译确认方法应用于 CompCert 编译器的优化验证中,通过用符号执行来说明代码段之间的语义等价性,实现对指令队列调度优化和跟踪调度优化的验证[15],通过对优化前后代码的控制流图 CFG 的静态分析实现对惰性代码优化[16]的验证以及基于符号计算实现对软件流水线[19]的翻译确认。有别于前人的是,在实现翻译确认算法时,Tristan 采用辅助证明系统 Coq 实现,确认器的程序、性质及形式化验证都采用同一语言描述,完全确保了翻译确认器的正确性。

此后,Barthe 将翻译确认方法用于对 CompCert 中基于 SSA 形式的中间编译过程及编译优化的验证[35],其确认器的实现也采用 Coq 实现。

4 翻译确认方法的主要研究方向

翻译确认方法自出现以来得到了广泛的研究与应用,已适用于多种编译器及编译优化,极大地提高了编译器的可信性。随着翻译确认方法的研究深入以及编译器的发展,翻译确认方法的研究呈现出多样化多角度的局面,目前翻译确认方法从不同的角度可以分为以下研究方向。

从使用对象来说,可以分为:1)面向翻译过程的翻译确认,其确认的对象是整个编译器或两种不同中间语言之间的编译过程,确认从源语言到目标语言的编译过程中的语义等价性。例如 Pnueli、Ryabtsev 以及 Tate 的工作;2)面向翻译优化的翻译确认,其确认的对象是编译过程中的局部优化,确认优化前和优化后代码的语义等价性,例如 Necula、Trsitan 等的工作。

从适用范围来说,可以分为:1)对于翻译确认的通用架构的构造,例如 Zuck 和 Pnueli 对翻译确认器 CVT 的研究工作,Barret 对 TVOC 的研究工作[25],Zaks 和 Pnueli 对确认器 CoVaC 的研究工作[26];2)用于实际编译器的通用确认算法的开发,例如 Pnueli,Necula (2000),Barret et al. (2005),Rinard (1999) 以及 Rival,他们依靠通用技术如符号执行、模型检验和定理证明等算法实现了对编译器的确认工作,这些算法能适用于大部分的程序转换,因此检查两段代码之间的语义等价性一般是不定的,这些确认器有着高复杂性可能会产生错误的警告;3)用于特定编译或优化的专用确认器,是针对某个特殊的优化或一组相关的优化的确认器。例如 Huang 针对寄存器分配的工作[20]、Tristan 和 Leroy 针对队列和跟踪调度的工作等,这些确认器基于有效的静态分析,因此更为正确和

完整(此分类引自文献[16,20])。

结束语　翻译确认是在编译时对编译器进行验证,发现编译错误的一种形式化方法,翻译确认的算法相对简单,且对编译器的具体实现不敏感,因此更易于实现对编译器的形式化验证又不影响编译器未来的优化完善,同时又不会对编译器的性能产生影响,能极大地提升编译器的可信性,同时又具备可重用性,因此在编译器验证、测试以及维护中都得到了广泛的应用。

翻译确认方法发展至今,相比于 Pnueli 提出概念之初,日趋成熟,同时已呈现出一些新的特点:

(1)算法更多样化,不拘泥于特定的语义框架或模型,将各种技术融入于翻译确认算法中,如模型检验、自动定理证明、数据流分析及符号执行等;

(2)适用范围更广泛,从对同构语言之间的编译到非同构语言之间的编译,从过程内优化的验证到过程间优化的验证,攥括了编译器的方方面面;

(3)部分确认器采用辅助证明系统实现,Analyzer 和证明检查器融为一体,更为完整和正确。

在构造可信编译器的过程中,翻译确认方法既可以单独使用,如 Ryabtsev 对 RTW 所做的工作[13],也可以和其他的形式化验证等方法有机地组合使用,例如 CompCert 编译器中使用自动定理证明对中间编译过程进行证明,同时也采用了翻译确认方法对软件流水线、指令调度等编译优化进行验证。翻译确认方法给编译器的可信性带来更大的保障,同时还提高了工作效率,相信其未来在构造可信编译器中将会发挥越来越重要的作用。

参 考 文 献

[1] Necula G C. Proof-Carrying Code[C]// POPL'97. ACM press, 1997:106-119

[2] Cimatti A, Giunchiglia F, Pecchiari P, et al. A Provably Correct Embedded Verifier for the Certification of Safety Critical Software[C]// Grumberg O, ed. Proc. 9th Intl. Conference on Computer Aided Verification (CA V'97). Lect. Notes in Comp. Sci., Springer-Verlag, 1997, 1254:202-213

[3] Pnueli A, Siegel M, Singerman E. Translation Validation[C]// Proc. 4th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98). 1998:151-166

[4] Pnueli A, Shtrichman O, Siegel M. Translation Validation for Synchronous Languages[C]// Larsen K G, Skyum S, Winskel G, eds. ICALP 1998. LNCS. Heidelberg: Springer, 1998: 1443: 235-246

[5] Pnueli A, Singerman E, Siegel M. Translation Validation: from SIGNAL to C[C]// Volume 1710 of LNCS State-of-the-Art Survey. Springer-Verlag, 1999:231-255

[6] Pnueli A, Shtrichman O, Siegel M. Translation validation: From DC+ to C[C]// Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods. Volume 1641 of Lect. Notes in Comp. Sci., 1998:137-150

[7] Pnueli A, Siegel M, Shtrichman O. The code validation tool (CVT)-automatic verification of a compilation process[J]. Int. Journal of Software Tools for Technology Transfer (STTT), 1999, 2(2):192-201

[8] Necula G C. Translation validation for an optimizing compiler [C]// Programming Language Design and Implementation 2000. ACM Press, 2000:83-95

[9] Zuck L, Pnueli A, Leviathan R. Validations of optimizing compliers[R]. Weizmann Institute of Science, 2000

[10] Zuck L, Pnueli A, Fang Y, et al. Voc: A methodology for the translation validation of optimizing compilers[J]. Jounal of Universal Computer Science, 2003, 9(3):223-247

[11] Zuck L, Pnueli A, Fang Y, et al. Translation and Run-Time Validation of Optimized Code[C]// Electronic Notes in Theoretical Computer Science. 2002

[12] Zuck L, Pnueli A, Goldberg B, et al. Translation and run-time validation of loop transformations[C]// Proceedings of the Run-Time Result Verification Workshop, Electronic Notes in Theoretical Computer Science (ENTCS). volume 70, 2002

[13] Ryabtsev M, Strichman O. Translation validation: From Simulink to C[M]// Computer Aided Verification. Springer, 2009

[14] Rival X. Symbolic transfer function-based approaches to certified compilation[C]// 31st symposium Principles of Programming Languages. ACM Press, 2004:1-13

[15] Tristan J-B, Leroy X. Formal verification of translation validators: A case study on instruction scheduling optimizations[C]// 35th symposium Principles of Programming Languages. ACM Press, 2008:17-27

[16] Tristan J-B, Leroy X. Verified Validation of Lazy Code Motion [C]// Programming Language Design and Implementation 2009. ACM Press, 2009:316-326

[17] Tate R, Stepp M, Tatlock Z, et al. Equality saturation: A new approach to optimization[C]// 36th Principles of Programming Languages. ACM, 2009:264-276

[18] Tristan J-B, Leroy X. Evaluating Value-Graph Translation Validation for LLVM[C]// 32nd ACM SIGPLAN conference on Programming language design and implementation. ACM Press, 2011:295-305

[19] Tristan J-B, Leroy X. A Simple, Verified Validator for Software Pipelining[C]// 37th symposium Principles of Programming Languages. ACM Press, 2010:83-92

[20] Tristan J-B. Formal verification of translation validators [D]. Universite Paris-Diderot-Paris VII, 2009

[21] Huang Yu-qiang, Childers B R, Soffa M L. Catching and identifying bugs in register allocation[C]// Static Analysis, 13th Int. Symp., SAS 2006. volume 4134 of Lecture Notes in Computer Science. Springer, 2006:281-300

[22] Kanade A, Sanyal A, Khedker U. A PVS based framework for validating compiler optimizations[C]// SEFM'06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods. IEEE Computer Society, 2006: 108-117

[23] Colby C, Lee P, Necula G C, et al. A certifying compiler for Java [J]. ACM SIGPLAN Notices, 2000, 35(5):95-107

[24] Chirica L M, Martin D F. An approach to compiler correctness [J]. ACM SIGPLAN Notices, 1975, 10(6):96-103

[25] Ngo V C, Talpin J-P, Gautier T, et al. Formal Verification of Synchronous Data-flow Compilers[R]. Project-Team ESPRESSO Research Report, March 2012:7921

[26] Rinard M, Marinov D. Credible compilation with pointers[C]// Workshop on Run-Time Result Verification. 1999

[27] Barret C W, Fang Yi, Goldberg B, et al. TVOC: A translation validator for optimizing compilers[C]// Computer Aided Verifi-

cation, 17th Int. Conf., CAV 2005. volume 3576 of Lecture Notes in Computer Science. Springer, 2005: 291-295

[28] Zaks A, Pnueli A. Covac: Compiler validation by program analysis of the cross-product[C] // FM 2008: Formal Methods, 15th International Symposium on Formal Methods. volume 5014 of Lecture Notes in Computer Science. Springer, 2008: 35-51

[29] Leroy X. Formal verification of a realistic compiler[M] // Communications of the ACM, 2009

[30] Leroy X. The Compcert verified compiler, software and commented proof[OL]. http://compcert.inria.fr/, Jan. 2010

[31] Fang Yi. Translation Validation of Optimizing Compilers[D]. New York University, 2005

[32] Goldberg B, Zuck L, Barret C. Into the loops: Practical issues in translation validation for optimizing compilers[C] // Proc. Workshop Compiler Optimization Meets Compiler Verification (COCV 2004). volume 132 of Electronic Notes in Theoretical Computer Science. Elsevier, 2005: 53-71

[33] Pnueli A, Zaks A. Translation validation of interprocedural optimizations[C] // Proceedings of the 4th International Workshop on Software Verification and Validation(SVV). 2006

[34] Stepp M, Tate R, Lerner S. Equality-Based translation validator for LLVM[C] // CAV'11: Proceedings of the 23rd International Conference on Computer Aided Verification. 2011

[35] Barthe G, Demange D, Pichardie D. A formally verified SSA-based middle-end-Static Single Assignment meets CompCert[C] // ESOP'12. 2012

[36] 何炎祥, 刘陶, 吴伟. 可信编译器关键技术研究[J]. 计算机工程与科学, 2010, 32(8): 1-6

(上接第 302 页)

表示使用 IP 地址随机测度、双向连接率、高端口法 3 个特征的方法。结果如表 2、表 3 所列。

表 2 3 种算法 10 次测试的检测率

次数	M_B	M_E	M_H
1	0.9	0.9	0.9
2	0.8	0.8	0.9
3	0.7	0.8	0.9
4	0.8	0.8	0.9
5	0.6	0.8	0.9
6	0.9	0.7	0.9
7	0.7	0.7	0.9
8	0.7	0.8	0.9
9	0.8	0.8	0.9
10	0.7	0.8	0.9

表 3 3 种算法 10 次测试的误报率

次数	M_B	M_E	M_H
1	0.15	0.025	0
2	0.075	0	0.025
3	0.075	0	0.025
4	0.15	0.025	0.025
5	0.1	0	0.025
6	0.075	0.075	0
7	0.15	0	0.025
8	0.15	0	0.025
9	0.1	0	0.025
10	0.075	0.025	0

从实验结果看出, IP 地址随机测度是比较理想的特征, 具有很高的检测率和较低的误报率。高端口特征与具体的实验环境和人为因素有关。实验中所有 P2P 客户端采用的是缺省配置, 监听端口均为高端口, 局域网中其他的应用也比较单纯, 主要运行一些常规的应用, 如浏览网页, 收发邮件等。而在实际更复杂的网络环境中, 可能存在 P2P 用户指定低端口为监听端口, 一些特殊的非 P2P 应用如 SQL 等也采用高端口_高端口等情况。因此, 高端口_高端口特征在这样一个局域网环境内有效并不能表明其在大规模的网络环境中就一定有效。

对算法误报率做进一步的测试, 容易造成误判的应用主要是一些服务器如 Dns 服务器、Mail 服务、视频服务器、Web 服务器等。对校园网的服务器网段进行了镜像, 服务器网段包括校园网 Dns 服务器、Email 服务器、Web 服务器和若干托管的服务器, 一共 15 台服务器。网段里没有使用 P2P 的主机, 构成了一个没有 P2P 的环境。同样抽样测试 10 次, 每次系统运行 2 分钟。误报率如表 4 所列。

表 4 10 次测试的误报率

次数	M_B	M_E	M_H
1	0.13	0	0.06
2	0.33	0.06	0.06
3	0.13	0	0.06
4	0.13	0	0.06
5	0.26	0.06	0.06
6	0.13	0	0.06
7	0.26	0.06	0.06
8	0.13	0	0.06
9	0.26	0.06	0.06
10	0.13	0	0.06

M_H 方法每次都有误报, 因为网段中某台服务器的监听端口为 8000, 所以 M_H 每次测试都产生了误报, 这也说明实际环境中使用 M_H 是很不准确的。M_B 方法 10 次检测结果不稳定, 波动比较大。进一步分析 M_B 方法每次都误报 Dns 服务器、Email 服务器, 这与单独实验是一致的, 对 Web 服务也产生了几次误报。Dns, Email 应用都表现出既有输入连接, 又有输出连接, 且连接数多, 与 P2P 的连接特征十分类似, 造成 M_B 方法误报率很高。M_E 方法较好, 误报率低, 且每次误报都是 Email 服务器, 可进一步调整门限。

结束语 P2P 网络中, 每个节点同时承担服务器和客户机的双重功能, 这是与 C/S 结构的根本区别, 对 P2P 主机远端地址分布进行了研究。由于资源随机分布在不同的 P2P 节点上, 导致了 P2P 主机连接的远端地址的均匀分布。据此, 提取了 P2P 主机的 IP 地址随机测度特征。实验表明该特征能有效地区分 P2P 主机和非 P2P 主机, 该特征计算简单, 适合实时处理。

参 考 文 献

[1] 鲁刚, 张宏莉, 叶麟. P2P 流量识别[J]. 软件学报, 2011, 12(6): 1281-1285

[2] 张琛, 王亮, 熊文柱. P2P 僵尸网络的检测技术[J]. 计算机应用, 2010, 6(2): 117-119

[3] 李鑫, 刘东林. 基于统计特征的 P2P 流量检测方法[J]. 计算机工程, 2010, 36(5): 114-116