可演化网络中移动代理的性能分析及优化方法

刘涛[1,2]，钱德沛[1,3]，黄泳翔[1]，王锐[3]，栾钟治[3]
（1.西安交通大学电子与信息工程学院，710049，西安；2.中国科学院深圳先进技术研究院，518055，广东深圳；
3.北京航空航天大学计算机学院，100191，北京）

摘要：针对以移动代理软件方式实现的可演化网络的性能问题，提出了一种快速激活及加载方法.通过对移动代理非执行转发、执行转发及执行各步骤的详细性能测量及分析，发现网络节点的执行环境为移动代理执行所做的准备工作时间消耗较大.该快速方法首先解开移动代理包中的所有程序代码并进行预先加载，在加载主程序时优先搜索已预先加载的代码，从而能够较快地找到程序代码.快速方法通过优化代码加载时的搜索顺序，有效地加快了可演化网络中移动代理的执行过程.测试结果表明，采用快速方法使典型的移动代理平均启动加载时间缩短了3.26 ms，网络节点的吞吐率提高了58%.

关键词：可演化网络；移动代理；性能优化
中图分类号：TP393 **文献标志码**：A **文章编号**：0253-987X(2010)12-0005-05

Performance Analysis and Optimization for Mobile Agents in Evolutionary Network

LIU Tao[1,2], QIAN Depei[1,3], HUANG Yongxiang[1], WANG Rui[3], LUAN Zhongzhi[3]
(1. School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China; 2. Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen Guangdong, 518055, China; 3. School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract: A novel fast launch algorithm is proposed to improve the performance of mobile agents in evolutionary networks. Analyses on the performance data of mobile agent's non-execution forwarding, execution forwarding, and detail execution steps show that the execution environment preparing process for mobile agents is the bottleneck. When the fast algorithm is executed, all codes in the mobile agent package will be extracted and loaded prior to the mobile agent's main program. Before a new code is loaded, the preloaded codes will be searched, and hence the code image can be located quickly. The mobile agent's execution forwarding can be improved by optimizing the searching order. Testing results show that the average loading time of the mobile agents is reduced by 3.26 ms and the throughput of network nodes is increased by 58%.

Keywords: evolutionary network; mobile agent; performance optimization

目前，互联网络的数据包转发大都是由特定硬件来实现的，一旦部署就难以改变，已经不能满足网络不断发展的新需求.研究人员探索了多种方式来提高网络对变化的适应能力，包括对网络硬件角色重定义而开发的软硬结合的网络节点[1]、支持对外接口的开放路由器[2]、支持协议更新的主动网络技术[3]、支持网络软件可以动态感知网络状态的可演化网络[4]等.以软件方式实现的可演化网络系统，采用移动代理技术来实现，具有较大的灵活性，但与硬件实现的路由器相比，不可避免地存在性能问题.ANTS[5]是一种基

收稿日期：2010-07-26. 作者简介：刘涛(1980—)，男，博士生；钱德沛(联系人)，男，教授，博士生导师. 基金项目：国家自然科学基金资助项目(90104022)；国家"863计划"资助项目(2006AA01A109，2009AA01Z144).

于移动代理的主动网络系统，它采用主动包动态执行的方式来实现新的网络协议，但由于每个主动包在转发时都经过了代码传输和代码执行过程，因此性能较低. PAN[6]是一种用 C 语言编写的软件主动网络系统，其代码在内核和用户态都需要执行程序，虽然其性能相对较高，但是具有很大的安全风险. 可演化网络的移动代理只在用户态以受控的方式执行，比 PAN 系统具有更好的安全性. Click[7]是一种软件方式实现的可编程网络路由器，性能较好，但是其代码在运行时并不能动态加载，扩展性受到限制. OpenFlow[8]通过在路由器上增加接口，可以通过连接新硬件的方式增强网络功能，但是仍然无法方便地随时部署自定义的网络功能，可扩展性受到硬件的限制. 可演化网络可以随时通过部署新的移动代理以增加系统功能，比 Click 和 OpenFlow 系统具有更大的灵活性，通过本文的性能优化后，可演化网络路由器的代理执行转发的吞吐率得到大大提高，在保证灵活性的情况下提高了网络性能.

本文详细分析可演化网络路由器与传统路由器在软件执行方式和性能上的区别，从改进代码动态部署执行效率入手，分析了移动代理的性能瓶颈，发现移动代理在代码加载方面具有较大的改进空间，提出了一种快速激活及加载方法，改善了可演化网络移动代理执行的性能，该方法对基于 JAVA 平台的移动代理系统具有普适参考价值.

1 移动代理执行过程原理

可演化网络的柔性来源于其可编程性，即可以动态地在网络节点上部署移动代理(Mobile Agent，MA)，扩展新的功能，实现新的协议，改变网络的行为. 移动代理的功能主要是对用户的传输任务和网络服务进行参数配置与管理，它在网络节点中得到资源并运行，并在节点上留下执行的结果. 移动代理可用来管理网络节点、部署新的网络服务以及为终端应用软件管理传输任务. 移动代理由移动代理程序、程序参数和负载 3 个部分组成. 程序参数包括移动代理程序的配置信息、用户验证信息、临时会话信息等. 移动代理的执行原理如图 1 所示，在可演化网络节点中的执行过程可以归纳为 7 个步骤.

(1)移动代理包的接收. 移动代理的包被管理器识别，导入网络节点的系统执行环境. 从接到移动代理包到开始处理的等待时间用 T_{in} 表示.

(2)移动代理包的解析. 系统执行环境解析移动代理包，识别移动代理程序、程序参数和负载，作相应

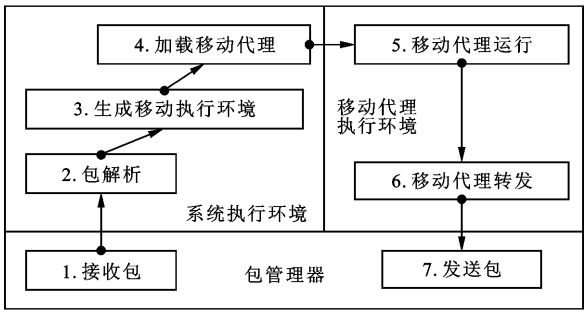

图 1 移动代理的执行原理

的验证和缓存. 包解析的时间用 T_{parse} 表示.

(3)生成移动代理执行环境. 系统执行环境根据移动代理包携带的认证信息和配置信息生成一个移动代理执行环境. 该执行环境被赋予一定的权限和资源. 生成并配置移动代理执行环境的时间用 T_{create} 表示.

(4)加载移动代理程序. 系统执行环境利用生成好的临时执行环境加载移动代理程序，加载程序的时间用 T_{load} 表示.

(5)移动代理程序执行. 移动代理程序在临时执行环境中启动其主线程，通过对临时执行环境的控制，可以限制移动代理的运行权限，防止移动代理破坏和滥用节点资源. 执行时间用 T_{run} 表示.

(6)移动代理转发排队. 移动代理执行完毕后，有可能会移动到下一个节点，需要重新对移动代理程序，程序参数和负载打包，并发送到它自定义的下一个节点. 这部分时间用 T_{queue} 表示.

(7)移动代理实际发送. 移动代理包通过包管理器排队后发送至网络，这部分时间用 T_{out} 表示.

设 T_{ma} 为一个移动代理通过可演化网络节点所需要的时间，可以得到

$$T_{\text{ma}} = T_{\text{in}} + T_{\text{parse}} + T_{\text{create}} + T_{\text{load}} + T_{\text{run}} + T_{\text{queue}} + T_{\text{out}}$$

这 7 部分时间可以分成被动性时间和主动性时间两类.

被动性时间是指接收、存储、转发一个包的时间，与传统网络设备中每个包所花费的转发时间类似，包括 T_{in}、T_{queue}、T_{out}. 另外，对于传统网络节点，有一个路由时间，即查找路由表的时间 T_{route}，IP 包通过网络节点的被动性时间为

$$T_{\text{ip}} = T_{\text{in}} + T_{\text{route}} + T_{\text{queue}} + T_{\text{out}}$$

主动性时间是指引入主动性而产生的额外的延时，包括包解析、执行环境生成、装载程序和程序执行的时间，即

$$T_{active} = T_{parse} + T_{create} + T_{load} + T_{run}$$

可演化网络节点和传统网络节点相比,转发移动代理和转发普通包的时间相差的正是主动性时间

$$\Delta T = T_{ma} - T_{ip} = T_{active} - T_{route}$$

由此可见,影响可演化网络节点性能的原因主要是引入了移动代理包解析、生成移动代理执行环境、装载移动代理程序和移动代理程序运行 4 个部分额外的时间开销.

2　移动代理性能瓶颈分析

本文研究小组在早期的研究中已经用 Java 技术实现了一个可演化网络原型系统[4]. 本文待测试的网络节点为一台双穴主机,其硬件配置 CPU 为 Pentium4 2.0 GHz,网卡速率为 100 Mb/s,发送机、测试网络节点、接收机三者利用网线直接相连. 在发送机上,首先将移动代理和长度不等的填充数据打包为移动代理包,总的包长度分别填充至 1 500～4 000 B,然后通过循环,将移动代理包发送至待测试的网络节点. 网络节点在接收移动代理包后将其转发至接收机. 接收机接收网络节点转发过来的移动代理包,并依次记录下时间戳. 在这种测试环境下,由于测试网络节点中各个移动代理执行之间不存在等待的时间间隔,可认为前一个移动代理的处理结束时间就是后一个移动代理的处理开始时间,因此相邻 2 个时间戳之差就表示单个移动代理转发的耗时,该耗时只与本节点的软硬件性能有关.

实验采用的移动代理为 Ping 移动代理,其主要功能是查找路由表,转发自身至下一个节点,其程序代码示意如下:

```
public class PingMA extends GeneralMA{
  public void real_run(){
  /*获得路由表*/
  Route r=(Route) Manager.getAPI("Route");
  /*取得下一跳地址*/
  IP d=r.getDefaultNextHop(dst);
  /*转发自身至下一跳*/
  sendCapsule(d);
  }
}
```

整个性能测试过程分为 3 部分:①宏观地测试被动性时间,移动代理程序并不真正执行,被当作普通数据包被转发,即测试移动代理非执行转发的性能;②宏观地测试主动性时间,即移动代理程序执行并转发的性能;③详细测试移动代理执行过程中主要耗时步骤的时间消耗.

2.1　非执行转发性能测试

移动代理的非执行转发是指将移动代理包当作普通数据包进行处理,并不执行其中的移动代理程序,这样可以测试出可演化网络节点中数据包管理与转发相关模块的性能. 实验记录下的不同长度移动代理非执行转发耗时的累积分布概率情况如图 2 所示,不同长度的移动代理包非执行转发的平均性能如图 3 所示.

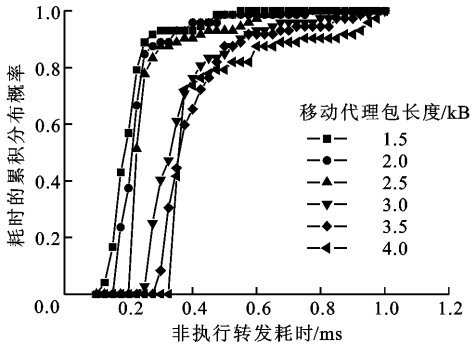

图 2　移动代理非执行转发耗时的累积分布图

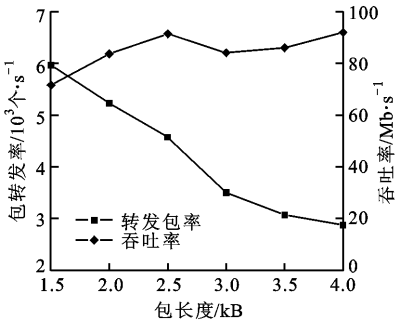

图 3　移动代理非执行转发的平均性能

从图 2 中可以看出,针对同样大小的包,占总量 80%的包的转发时间基本一样,转发性能较为稳定,表现为耗时的累积分布概率在 0.1～0.8 之间的曲线比较陡,而有少量包由于排队等待,造成转发时间较长,如图中累积分布概率在 0.9 以上曲线的耗时较长. 从图 3 中可以看出,随着包逐渐变长,包的平均转发时间变长,包转发率逐渐下降,吞吐率则稳定在 70 Mb/s 至 90 Mb/s 之间,接近实验平台 100 Mb/s 网卡的极限速度. 由此可以得知,原型系统的数据包管理部分虽然额外为移动代理包添加了排队、路由等操作,但是对传统网络系统的包传输影响较小,即对包转发的被动性时间影响很小.

2.2　执行转发性能测试

移动代理执行转发是指移动代理在节点上先执

行再转发，整个转发时间包括了移动代理的执行时间，即主动性时间. 测试同样采用移动代理 Ping 来进行. 实验记录下的不同长度移动代理 Ping 的执行转发耗时的累积分布概率情况如图 4 所示，不同长度的移动代理执行转发的平均性能如图 5 所示.

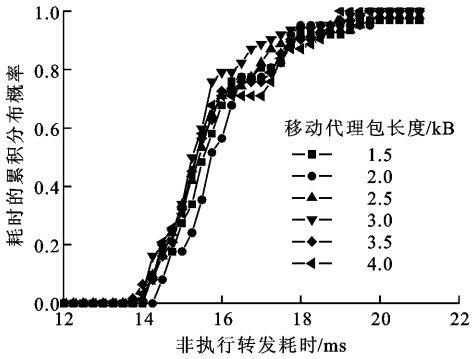

图 4 移动代理执行转发耗时的累积分布图

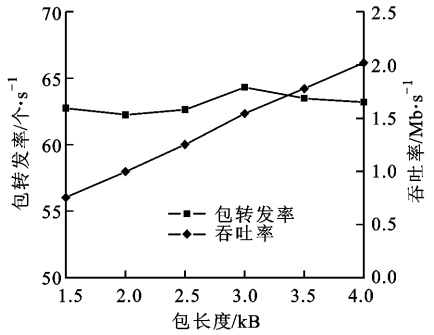

图 5 移动代理执行转发的平均性能

从图 4 中可以看到，移动代理由于在转发过程中额外添加了程序代码执行等复杂的过程，性能不够稳定，测得的转发时间分布较宽，在图中表现为耗时的累积分布比例在 0.1 至 0.9 之间的曲线较为平缓. 从图 5 中可看出，移动代理执行转发的性能与包长的关系不大，包转发率均为 63 个/s，吞吐率远远小于网络接口的物理限制. 其主要原因在于移动代理执行转发耗时远大于非执行转发，如包长为 4 kB 时，执行转发的吞吐率约为 2 Mb/s，而非执行转发的吞吐率约为 91 Mb/s，性能相差 45 倍. 包长较小时，性能相差可达 100 倍左右. 由此可见，移动代理执行转发时传统的数据包转发步骤耗时较少，其主要性能瓶颈在于代码执行等步骤.

2.3 转发各步骤性能测试与分析

在详细测试移动代理转发各步骤的时间消耗实验中，采用了两种移动代理程序，一种为 Ping，另一种为 Ping+Comp，这种移动代理在 Ping 的基础上多加了一部分计算时间，即 300 次字符串相加操作，可以用来模拟具有较大计算量的移动代理.

对这两种移动代理执行过程的主要耗时的步骤进行分别计时，表 1 给出一次移动代理转发中主动性时间的平均测试结果.

表 1 主动性时间比较

移动代理	T_{parse}/ms	T_{create}/ms	T_{load}/ms	T_{run}/ms	T_{active}/ms
Ping	0.04	4.15	9.39	0.14	13.72
Ping+Comp	0.04	4.15	9.39	7.67	21.25

从表 1 中可以看出，由于两种移动代理的程序大小相差很少，因此其包解析、生成执行环境、程序装载的时间基本一样. 另外，由于两者的计算任务差别较大，因此在执行时间上存在较大的差距. 对于计算量较小的移动代理程序（例如 Ping），执行时间较短，T_{create} 和 T_{load} 占据了主动性时间的绝大部分，约 98.7%. 对于较为复杂的移动代理程序（例如 Ping+Comp），虽然其执行时间变长，然而 T_{create} 和 T_{load} 仍然占据了主动性时间的大部分，约 63.7%. 大多数移动代理作为可演化网络的主要管理与控制手段，单次执行时间并不会太长. 从上述两个移动代理的测试结果可以看出，为移动代理程序运行所做作的准备工作，即生成执行环境和移动代理加载，占据了大部分时间开销，是可演化网络节点的性能瓶颈所在. 为了提高节点的性能，应该首先对这部分进行优化.

3 快速激活及优化加载方法

针对优化移动代理加载性能的需求，本文提出了一种移动代理快速激活及优化加载方法. 传统的 Java 类加载采用按需加载方法，即在程序需要使用一个未加载的类时，要进行代码的查找和加载的动作. 在加载一个新的 Java 类时，首先要搜索当前环境上下文中的类加载器及其父加载器的代码仓库，需要进行大量的查询操作，效率较低. 从上述实验环境中的测试结果（表 1）可知，对只含有一个类的简单的移动代理进行加载就需要花费 9.39 ms. 对 Java 类加载过程的分析可知，加载过程的效率与 Java 类加载时的查找顺序密切相关. 在可演化网络中，需要加载的移动代理类的代码就包含在新下载的移动代理包中，而不是在代码仓库中，对父加载器代码仓库进行的查找操作是无用的，浪费了大量时间，因此具有较大的优化空间.

本文提出的快速激活及优化加载方法流程如图 6 所示，具体的伪代码如图 7 所示. 在移动代理包经过解包和分析之后，将移动代理包中的所有程序代码

加入当前类加载器的本地代码库中. 在加载移动代理类及同属同一个 Java 包的所有类时,优先查找本地代码库,这样移动代理的主程序及其相关类的代码均可以直接在本地代码库中找到并进行加载,省略了查找父加载器甚至更远的根加载器的查找步骤. 对于非同一包所属的 Java 类,仍采用传统的类查找和加载方法,对于无法加载的类,则向代码服务器进行远程查找:若找到,则从代码服务器下载代码,然后完成加载;若找不到,则抛出异常,并记录异常日志.

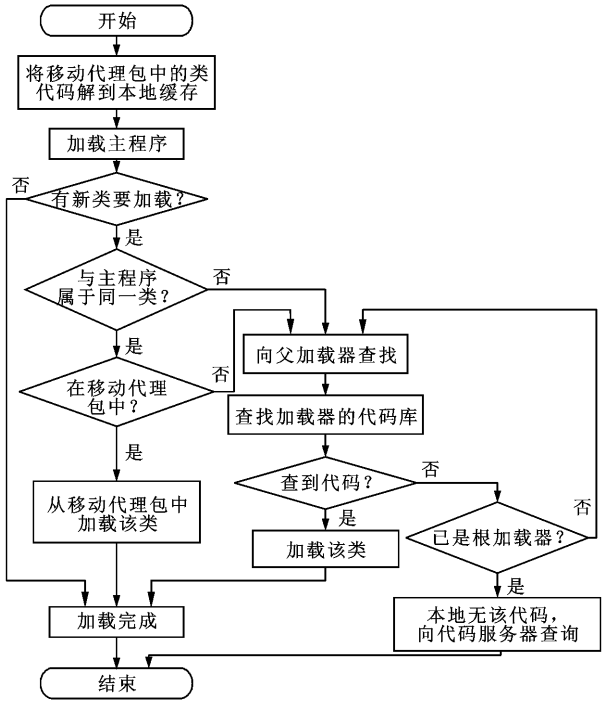

图 6　快速激活及优化加载方法流程

与未优化相比,本文方法在开始时建立了移动代理包中所有代码的列表,由于移动代理包中的代码数量有限,因此所消耗的内存并不会很大. 对于主程序和该列表中的所有代码,只需一次查找就可以找到. 对于属于同一 Java 包,而又不在该列表中的代码,需要增加对该列表的遍历操作,但是由于该表长度有限,对这部分代码加载的性能影响有限. 对于剩下的其他 Java 类,需要判断该类是否与主程序属于同一包,增加的这部分判断时间基本可以忽略不计.

在对移动代理加载步骤进行优化后,同样利用 Ping 作为实验对象,测试其执行转发的性能数据,优化后的移动代理 Ping 的转发性能如图 8 所示. 这种快速的激活及加载方法使移动代理主程序的平均启动加载时间减少了 3.26 ms,而且移动代理程序在后续执行时每次对其所依赖代码的加载性能也得到了相应的优化. 从图 8 中可以看出,优化后,移动代理执行转发的性能提高了,其包转发率可达 100 个/s,比未优化时性能提高了 58%.

```
Input: M: MAClassName, P: MAPackage
LoadClass (M):
L = ClassExtract (MAPackage);
DependClass = Depend (MAClassName);
for every C in DependClass do
  if C is Java Standard Class then
    SystemLoader.LoadCode (C)
  else if SamePackage (C, M)
    and (C in L) then
    thisLoader.LoadCode (C, P);
  else
    Delegate to ParentClassLoader
  end if
  if error then
    Throw Exception (Dependency Not Found!);
  end if
 end for
 if M in L then
  thisLoader.LoadCode (M, P);
 else
  Delegate to ParentClassLoader
end if
```

图 7　快速激活及优化加载方法的伪代码描述

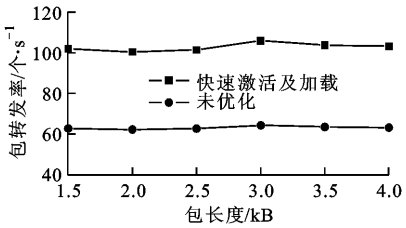

图 8　快速激活及优化加载方法的性能

4　结论及进一步工作

可演化网络是一种可动态感知网络状态变化的网络结构,可以更好地适应复杂的网络环境,其原型系统采用动态加载的结构和移动代理技术来实现. 本文在详细的性能测试及分析的基础上,针对代码装载的性能瓶颈问题,设计了快速激活及加载方法. 该方法通过优化代码加载时的查找顺序,能更快更准地找到正确的代码,缩短了移动代理的启动时间,有效地优化了可演化网络的性能. 下一步的工作要研究如何通过重用执行环境等方式进一步优化性能,并在移动网络下研究移动代理对提高网络性能的作用.

本文提出的方法对基于 Java 平台的移动代理系统具有普适性.

参考文献：

[1] MARTIN C, TEEMU K, DAEKYEONG M, et al. Rethinking packet forwarding hardware [EB/OL]. (2008-10-12)[2010-05-20]. http://conferences.sigcomm.org/hotnets/2008/papers/1.pdf.

[2] JEFFREY M, PRAVEEN Y, JEAN T, et al. API design challenges for open router platforms on proprietary hardware [EB/OL]. (2008-10-12)[2010-05-20]. http://conferences.sigcomm.org/hotnets/2008/papers/2.pdf.

[3] YAMAMOTO L, TSCHUDIN C. Experiments on the automatic evolution of protocols using genetic programming [C]// Proceedings of 2nd Workshop on Autonomic Communication. Berlin, Germany: Springer, 2006: 13-28.

[4] 刘涛，钱德沛，王锐，等. 一种可演化网络的研究与实践[J]. 西安交通大学学报，2008，42(10)：1193-1203.
LIU Tao, QIAN Depei, WANG Rui, et al. Research and practice on evolutionary network [J]. Journal of Xi'an Jiaotong University, 2008, 42(10): 1193-1203.

[5] DAVID W, JOHN G, DAVID T. Ants: a toolkit for building and dynamically deploying network protocols [C]// Proc of IEEE OPENARCH 98. Piscataway, NJ, USA: IEEE, 1998: 117-129.

[6] ERIK N, STEPHEN G, FRANS K. Pan: a high performance active network node supporting multiple mobile code systems [C]// Proc of IEEE OPENARCH 99. Piscataway, NJ, USA: IEEE, 1999: 78-89.

[7] EDDIE K, ROBERT M, BENJIE C, et al. The click modular router [J]. ACM Transactions on Computer Systems, 2000, 18(3): 263-297.

[8] NICK M, TOM A, HARI B, et al. OpenFlow: enabling innovation in campus networks [J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[本刊相关文献链接]

一种新的估计模数转换器积分非线性误差的直方图方法. 西安交通大学学报，2009，43(8)：59-63.

一种新型高阶两通道时间交织 ΣΔ 调制器系统结构. 西安交通大学学报，2008，42(8)：996-1000.

流水线模数转换器的一种数字校准技术. 西安交通大学学报，2008，42(8)：991-995.

流水线模数转换器中高速低功耗开环余量放大器的设计. 西安交通大学学报，2008，42(6)：751-755.

一种新型级数 Δ 调制器系统结构. 西安交通大学学报，2008，42(12)：1541-1545.

（编辑 刘杨）