# LWIP 中零拷贝技术的研究与应用

赵成青,李宥谋,刘永斌,王　涛
(西安邮电大学,陕西 西安 710000)

**摘　要**:LWIP 是一种轻量级的 TCP/IP 协议栈。在运行过程中占用少量的资源,主要应用在低端的嵌入式系统。文中从物理层到应用层,分三个层次分析了 LWIP 协议栈的数据传递过程。分别是物理层到协议栈内部的数据传递过程、协议栈内部本身的数据传递过程、协议栈和外部应用程序数据的传递过程。而数据在协议栈内部传递时,通过 pbuf 缓冲包在各层之间传递数据包的地址指针已经实现了数据在协议栈内部各层之间的零拷贝传递。提出了在物理网卡和 LWIP 协议栈传递数据、外部应用程序和 LWIP 协议栈传递数据时的改进方法,避免了数据的两次拷贝,从而实现了数据从物理层直达应用层,提高了系统的传输效率和并发性能。测试结果表明,数据传输速率从未优化的 2.04 MB/s 提升到 9.8 MB/s,已接近网卡性能极限。

**关键词**:分层;内存映射;指针传递;零拷贝;IPC 方式
**中图分类号**:TP216　　**文献标识码**:A　　**文章编号**:1673-629X(2018)07-0182-05
doi:10.3969/j.issn.1673-629X.2018.07.039

# Research and Application of Zero Copy Technology in LWIP

ZHAO Cheng-qing,LI You-mou,LIU Yong-bin,WANG Tao
(Xi'an University of Posts and Telecommunications,Xi'an 710000,China)

**Abstract**:LWIP,a lightweight TCP/IP stack,occupies a small amount of resources during the operation and is mainly used in low-end embedded systems. From the physical layer to the application layer,the data transfer process of the LWIP protocol stack is analyzed in three levels which are the data transfer process from the physical layer to the protocol stack,the data transfer process inside the protocol stack,and the data transfer between the protocol stack and the external application program. When the data is passed inside the protocol stack,the address pointer of the data packet is transmitted between the layers through the pbuf buffer packet,and the data has been transmitted by zero copy between the layers of the protocol stack. We propose an improved method to transfer data between the physical network card and the LWIP protocol stack,the external application program and the LWIP protocol stack,so as to avoid the two copy of the data. The system realizes the direct application of data from the physical layer to the application layer,and improves the transmission efficiency and the concurrent performance of the system. Test shows that the rate of data transmission has been optimized from 2.04 MB/s to 9.8 MB/s,which is close to the network card performance limit.

**Key words**:layered;memory mapping;pointer passing;zero copy;IPC mode

## 0　引　言

LWIP 是瑞典计算机科学院(SICS)的 Adam Dunkels 开发的用于嵌入式系统的开源 TCP/IP 协议栈[1]。LWIP 的含义是轻量级的 TCP/IP 协议,专注于减少资源消耗。嵌入式网络传输系统由于成本资源的限制,往往采用简化的 TCP/IP 协议。文中通过研究、分析常用的嵌入式网络协议栈 LWIP 的结构,在物理层和应用层提出了提高系统传输效率的改进方法。

在小型嵌入式系统中,LWIP 的实现基于上层协议已明确知道下层协议所使用的数据结构的特点[2]。它会假设各层间的部分数据结构和实现原理在其他层是可见的。在数据包递交过程中,各层协议可以通过指针直接指向数据包中其他层次的字段。所以上层可直接使用取地址计算得到下层中的数据,这不仅使整个协议栈对数据包的操作更加灵活,而且避免了 LWIP 协议栈内部数据递交时的复制。但是,这仅仅

**收稿日期**:2017-07-07　　**修回日期**:2017-11-15　　**网络出版时间**:2018-02-24
**基金项目**:陕西省重大科技创新专项资助项目(2010ZKC02-08)
**作者简介**:赵成青(1990-),男,硕士研究生,研究方向为嵌入式系统开发与设计;李宥谋,教授,研究方向为集成电路设计、嵌入式系统开发与设计。
**网络出版地址**:http://kns.cnki.net/kcms/detail/61.1450.TP.20180224.1521.066.html

是在 LWIP 协议栈内部实现数据的零拷贝。在物理网卡向协议栈传递数据时和协议栈向应用程序传递数据时,还是存在两次消耗较大的数据拷贝过程。所以,文中提出在网卡接收数据时让 LWIP 内核存储区指针直接指向物理网卡的寄存器地址的方法,避免了物理网卡数据到 LWIP 协议栈缓冲区数据的拷贝[3]。在应用层,提出了利用 μcos 操作系统的邮箱机制,避免了多个外部应用程序和协议栈内核交互时的数据拷贝[4],从而实现了从物理层到应用层真正的数据零拷贝,并且提高了系统的并发性。

## 1　物理网卡到 LWIP 协议栈数据的传递

在嵌入式系统中应用比较广泛的是 MicroChip 公司的 ENC28J60 网卡。在网卡驱动函数接收数据包时,网卡驱动函数首先向网卡发送数据包传送指令,此时网卡会把 BNRY(边界寄存器)处的一个网卡格式的数据包数据一次性全部发送到 DMA 端口,此时网卡驱动函数会在 DMA 端口读取所有字节数据后,网卡会自动将 BNRY(边界寄存器)值调整为下一个数据包地址,以准备下一次读取所有字节数据。然后将接收到的数据封装成 LWIP 熟悉的格式并且写到缓冲区中,这个过程涉及到数据到拷贝[5]。在 ENC28J60 网卡中接收缓冲器由一个硬件管理的循环 FIFO 构成。ERXST 表示接收缓冲区起始地址,ERXNDH 表示接收缓冲区结束地址[6]。如图 1 所示,通过把网卡相关的寄存器映射到 LWIP 内核内存空间,就可以直接对网卡寄存器进行操作,避免了物理网卡到 LWIP 内核空间的数据拷贝。然后封装成 LWIP 内核能够识别的 pbuf 类型的数据包结构,在 LWIP 内核中由 low_level_input( ) 函数完成这个功能。通过 ehternetif_input( ) 函数解析该数据包的类型,然后将该数据包指针递交给相应的上层。

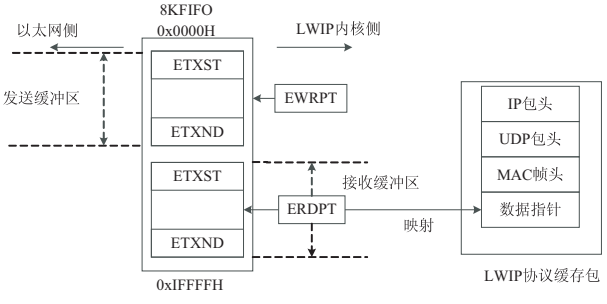



图 1　网卡内部寄存器指针映射

## 2　LWIP 协议层间数据传递

在网卡接收数据时,需要申请一个数据包,然后将网卡中的数据填入数据包中。发送数据包时,协议栈的某层中会申请一个 pbuf,并将相应的数据装入到数据区域,同时相关的协议首部信息也会被填写到 pbuf 的预留数据区域中[7]。数据包申请函数有两个重要参数,一个是想申请的数据包 pbuf 类型,另一个重要参数是该数据包是在协议栈中哪一层被申请的,分配函数会根据这个层次的不同,在 pbuf 数据区域前为相应的协议预留出首部空间,这就是 offset 值。总的来说,LWIP 定义了四个层次,当数据包申请时,所处的层次不同,会导致预留空间的 offset 值不同[8]。层次定义时通过一个枚举类型 pbuf_layer:

```
Typedef enum{
PBUF_TRANSPORT,//传输层
PBUF_IP,//网络层
PBUF_LINK,//链路层
PBUF_RAW,//原始层
}pbuf_layer;
```

上面的结构体中除了定义枚举类型 pbuf_layer 来表示各个网络协议层的名称外,还定义了两个宏:PBUF_TRANSPORT_HLEN 和 PBUF_IP_HLEN。前者是典型的 TCP 报文首部长度,而后者是典型的不带任何选项字段的 IP 首部长度。代码如下所示:

```
Switch(layer){
Case PBUF_TRANSPORT:
Offset = PBUF_LINK_HLEN + PBUF_IP_HLEN + PBUF_TRASNSPORT_HLEN;
Case PBUF_IP:
Offset=PBUF_LINK_HLEN+PBUF_IP_HLEN;
Case PBUF_LINK:
Offset=PBUF_LINK_HLEN;
Case PBUF_LINK:
Offset=0;
```

在 LWIP 的数据包管理函数 pbuf. c 中,首先根据数据包申请时传入的协议层参数,计算需要在 pbuf 数据区签预留的长度 offset 值,然后根据 pbuf 的类型进行实际申请。Pbuf_pooll 类型申请最复杂[9],因为可能需要几个 pool 连接在一起,以此来满足用户的空间需求。

限于篇幅,对 LWIP 内存分配机制不做深入研究;pbuf_ref 和 pbuf_rom 类型申请最简单,它们只是在内存 MEMEP_PBUF 中分配一个 pbuf 结构空间,然后初始化相关字段,注意这两种类型的 payload 指针需要用户自行设置,通常在调用完函数 pbuf_alloc 后,调用者需要将 payload 指向某个数据区。

在原始层以太网驱动中:

```
P=pbuf_alloc(PBUF_RAW,recvlen,PBUF_RAM);
```

这个调用语句申请了一个 PBUF_RAM 类型的 pbuf,且其申请的协议层为 PBUF_RAW,所以 pbuf_alloc 函数不会在数据区前预留出任何首部空间;通过使用 p->payload,就可以实现对 pbuf 中数据区的读取或

者写入操作了。

在传输层 TCP 层：

```
P=pbuf_alloc(PBUF_RAW,recvlen,PBUF_RAM);
```

它告诉数据包分配函数使用 PBUF_RAM 类型的 pbuf，且数据前应该预留一部分的首部空间。由于这里是 PBUF_TRANSPORT 层，所以预留空间将有 54 个字节，即 TCP 首部长度的 20 个字节、IP 数据包首部长度的 20 个字节以及以太网帧首部长度的 14 字节。当数据包往下层递交，各层协议就直接操作这些预留空间的数据，以实现数据首部的填写，这样就避免了数据的拷贝。

## 3 LWIP 软件与用户程序间的数据传递

### 3.1 用户缓冲数据结构

协议栈 API 实现时，也为用户提供了数据包管理函数，可以完成数据包内存申请、释放、数据拷贝等任务。无论是 UDP 还是 TCP 连接，当协议栈接收到数据包后，会将数据封装在一个 netbuf 中，并递交给应用程序[10]。在发送数据时，不同类型的连接将导致不同的数据处理方式。对于 TCP 连接，内核会根据用户提供待发送数据的起始数据和长度，自动将数据封装在合适的数据包中，然后放入发送队列；对于 UDP，用户需要手动将数据封装在 netbuf 中，通过调用发送函数，内核直接发送数据包中的数据段。

应用程序使用 netbuf 结构来描述、组装数据包，该结构只是对内核 pbuf 的简单封装，是用户应用程序和协议栈共享的。外部应用程序可以使用该结构来管理发送数据、接收数据的缓冲区。netbuf 是基于 pbuf 实现的，其结构如以下代码所示：

```
Struct netbuf{
Struct pbuf * p, * ptr;
Ip_addr_t * addr;
U16_t port;
}
```

其中，netbuf 相当于一个数据首部，保存数据的字段是 p，它指向 pbuf 链表首部，ptr 指向链表中的其他位置，addr 表示 IP 地址，port 表示端口号。

netbuf 是应用程序描述待发送数据和已接收数据的基本结构，引入 netbuf 结构看似会让应用程序更加繁杂，但实际上内核为应用程序提供了 API，通过共享一个 netbuf 结构（如图 2 所示），两部分 API 就能实现对数据包的共同处理，避免了数据拷贝。

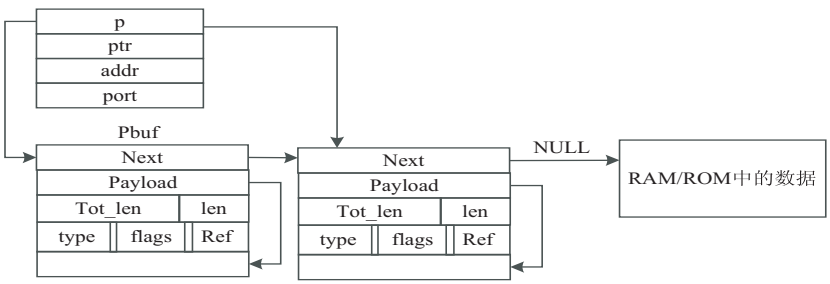



图 2 用户缓冲区结构

### 3.2 操作数据缓冲区指针

与 BSD 相同，LWIP 协议栈 API 也对网络连接进行了抽象。但它们之间的抽象存在一定的差别：BSD 实现了更高级别的抽象，用户可以像操作文件那样来操作一个网络连接；LWIP 中，API 只能实现较低级别的抽象，用户操作的仅仅是一个网络连接，而不是文件。在 BSD 中，应用程序处理的网络数据都处于一片连续的存储区域中，可以使用户对数据的处理更加方便。在 LWIP 中，若 API 使用上述数据存储机制可能会导致很大的缺陷，因为 LWIP 中网络数据都存储在 pbuf 中，如果要实现存储在连续的存储区的话，需要将所有 pbuf 数据拷贝到这个连续的存储中，这将造成数据的拷贝。为了避免数据拷贝以后再递交给用户，需要直接操作 pbuf 的一些方法，而 LWIP 中恰恰提供了这些方法。比如通过 netbuf_next( ) 可以修改数据指针指向下一个数据段，如果返回值为 0，表示 netbuf 中还存在数据段；大于 0 说明指针已经指向 netbuf 中的最后一个数据段了，小于 0 表明 netbuf 中已经没有数据段了。当用户未调用 netbuf_next( ) 函数的情况下，ptr 和 p 都默认指向第一个 pbuf。通过 netbuf_next( ) 对协议栈和应用程序共同缓冲区指针的调整和读取，避免了应用程序和数据以及内核栈的拷贝。

## 4 应对多个外部应用程序的指针传递

在单独运行 LWIP 时，用户应用程序和协议栈内核处于同一进程中，用户程序通过回调的方式进行。这样，用户程序和协议栈内核出现了相互制约的关系，因为用户程序执行的时候，内核一直处于等待状态，内核需要等待用户函数返回一个处理结果再继续执行。如果用户执行计算量很大，执行时间很长，则协议栈代码就一直得不到执行，协议栈接收，处理数据包效率会受到直接的影响。最严重的结果是，如果发送方速度很快，则协议栈会因为来不及处理而出现丢包的情况。

为了设计多进程外部应用程序，将 LWIP 移植到

μcos 操作系统下,让 LWIP 内核作为操作系统的一个任务运行[11]。LWIP 协议栈设计时,提供了协议栈与操作系统之间函数的接口。协议栈 API 由两部分组成。一部分提供给应用程序,一部分提供给协议栈内核。应用程序和协议栈内核通过进程间通信机制进行通信和同步[12]。使用到的进程通信机制包括了以下三种[13]:

(1)邮箱,例如内核邮箱 mbox、连接上接收数据的邮箱 recvmbox;

(2)信号量,例如 op_completed,用于两部分 API 同步;

(3)共享内存,例如内核消息结构 tcp_msg、API 消息内容 api_msg 等[14]。

两部分 API 间的关系如图 3 所示。API 设计的主要思想是让应用程序成为一个单独的进程;而协议栈也成为一个单独的进程。用户进程只负责数据的计算等其他工作,协议栈进程仅仅负责通信工作。两部分进程之间使用三种 IPC 方式中的邮箱和信号量集,内核进程可以直接将数据递交到应用程序邮箱中,然后继续执行,不必阻塞等待,邮箱对于应用程序来说就像一个输入队列,提高了系统的实时性[15]。

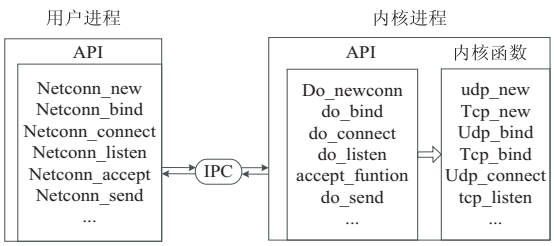


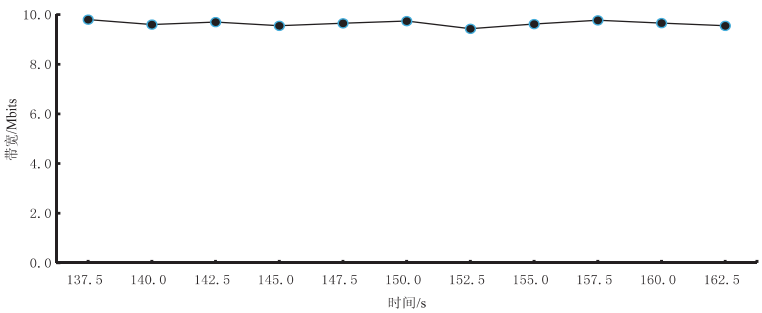

图 3　两部分独立进程间的通信

全局邮箱 mbox 在协议栈初始化时建立,用于内核进程 tcpip_thread 接收消息。内核进程通过共享内存的方式与协议栈的其他各个模块进行通信,它从邮箱中获得的是一个指向消息结构的指针。函数 tcp_input 在内存池中为系统消息结构申请空间,并根据消息类型初始化结构中的相关字段,把内核消息封装在 tcp_msg 结构中,最后将消息投递到系统邮箱中等待内核进程 tcpip_thread 处理。tcpip_thread 使用从邮箱中获得的指针指定到对应内存地址处读取消息内容,从而避免了两个进程间通信的数据的拷贝。

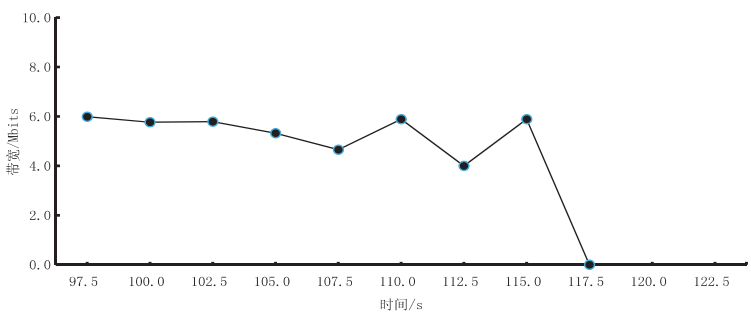
## 5　性能测试对比及其应用

在局域网内,对 ARM 开发板 STM32F103VET6-EV 上基于无操作系统和移植了 μcos 操作系统的 LWIP 两种方法编写的 UDP 服务器进行数据吞吐能力的测试,以此来估算网卡及整个板子的网络处理性能及对比无操作系统模拟层和在操作系统模拟层下编写的 UDP 服务器性能的差别。

在 Windows 主机上运行 iperf 软件来测试服务器的数据吞吐能力。如图 4(a)所示,在软件上选择 UDP 协议,设置好服务器 IP 地址(192.168.1.230)和端口号(5000)后,单击 start iperf,软件开始对服务器性能进行测试。从图 4(a)可以看出,服务器的上下行带宽都可以维持在 9 800 kb/s 左右,很接近 ENC28J60 网卡的处理值上线 10 M/s。在操作系统模拟层下基于 LWIP 零拷贝技术编写的 UDP 服务器,板子的网络处理性能达到最优。从图 4(b)可以看出,基于无操作系统模拟层下编程的服务器在客户端连续发送大量数据




(a)




(b)

图 4　UDP 性能测试

时导致丢包情况,严重情况下甚至出现死机的情况。

## 6 结束语

综上所述,在应对多个外部应用程序的情况下,无操作系统模拟层的 UDP 服务器编程,虽然避免了数据的拷贝,但是无法应对多个外部应用程序。所以将 LWIP 移植到 μcos 操作系统下,不仅减少了内存开销,而且能够应对多个外部应用程序。文中的研究成果已经成功应用于嵌入式网管系统项目并实际运行,不仅提高了基于 STM32 平台 μcos 操作系统下测量仪器代理模块的传输效率,提高了系统的实时性,而且节约了内存开销。

**参考文献:**


[1] CAROFIGLIO G,MUSCARIELLO L. On the impact of TCP and per-flow scheduling on internet performance[J]. IEEE/ACM Transactions on Networking,2012,20(2):52-56.

[2] SHERWANI S A,KHIYAL M S H. Real-time scheduler for transport protocols[J]. Information Technology Journal,2007,6(3):376-379.

[3] LI Jinlei,ZHENG Wengang,SHEN Changjun,et al. Application of modbus protocol based on μC/TCPIP in water saving irrigation in facility agricultural[C]//International conference on computer and computing technologies in agriculture. [s. l.]:[s. n.],2014.

[4] SCHIEPEK G, AICHHORN W. Real-time monitoring of psychotherapeutic change processes[J]. Psychotherapie,Psychosomatik,Medizinische Psychologie,2013,63(1):39-47.

[5] 陈 实,武 杰. 一种基于以太网的嵌入式数据传输速率优化方法研究[J]. 微型机与应用,2015,34(4):64-66.

[6] 张庆辉,马延立. STM32F103VET6 和 ENC28J60 的嵌入式以太网接口设计[J]. 单片机与嵌入式系统应用,2012(9):23-25.

[7] 张 齐,劳炽元. 轻量级协议栈 LWIP 的分析与改进[J]. 计算机工程与设计,2010,31(10):2169-2171.

[8] 蔡雄飞,王新华,郭淑琴. 嵌入式 TCP/IP 协议 LWIP 的内存管理机制研究[J]. 杭州电子科技大学学报,2012,32(4):118-121.

[9] 付晓军,夏应清,何 轩. 嵌入式 LWIP 协议栈的内存管理[J]. 电子技术应用,2006,32(3):56-58.

[10] 李茂正. 嵌入式系统中实现网络协议[D]. 南京:南京大学,2011.

[11] 苏勇辉. 基于 ARM 微处理器 TCP/IP 协议栈 LwIP 实现[J]. 国外电子测量技术,2009,28(10):76-78.

[12] MOLAY B. Unix/Linux 编程实践教程[M]. 杨宗源,黄海涛,译. 北京:清华大学出版社,2004:464-475.

[13] LABROSSE J J. 嵌入式实时操作系统 μC/OS-II[M]. 邵贝贝,译. 北京:航空航天大学出版社,2012:143-150.

[14] STEINKE R C,NUTT G J. A unified theory of shared memory consistency[J]. Journal of the ACM,2004,51(5):800-849.

[15] 任 哲. 嵌入式实时操作系统 μC/OS-II 原理及应用[M]. 北京:航空航天大学出版社,2005:133-139.


(上接第 181 页)


网络性能比较[J]. 中国卫生统计,2013,30(2):173-176.

[4] 卢辉斌,李丹丹,孙海艳. PSO 优化 BP 神经网络的混沌时间序列预测[J]. 计算机工程与应用,2015,51(2):224-229.

[5] 游丹丹,陈福集. 基于改进粒子群和 BP 神经网络的网络舆情预测研究[J]. 情报杂志,2016,35(8):156-161.

[6] 张宝堃,张宝一. 基于 BP 神经网络的非线性函数拟合[J]. 电脑知识与技术,2012,8(27):6579-6583.

[7] BUSCEMA M. Back propagation neural networks[J]. Substance Use & Misuse,1998,33(2):233-270.

[8] 范广坡,余学飞,卢广文,等. 改进 PSO-BP 算法的压力导丝温度及非线性补偿研究[J]. 自动化仪表,2016,37(6):16-20.

[9] 崔东文. 多隐层 BP 神经网络模型在径流预测中的应用[J]. 水文,2013,33(1):68-73.

[10] JIAN Huajiang,BU Yunsheng,LI Xionggong,et al. PSO algorithm based task allocation in dynamic virtual enterprise[J]. Advanced Materials Research,2011,189-193:2572-2576.

[11] 朱小明,张慧斌. PSO 算法的稳定性分析及算法改进[J]. 计算机科学,2013,40(3):275-278.

[12] 涂娟娟. PSO 优化神经网络算法的研究及其应用[D]. 镇江:江苏大学,2013.

[13] LIU Lilan, HU Rongsong, HU Xiangping, et al. A hybrid PSO-GA algorithm for job shop scheduling in machine tool production[J]. International Journal of Production Research, 2015,53(19):5755-5781.

[14] 梅金平,张士兵,王海莲. 基于 PSO-GA 算法的多用户 OFDM 系统资源分配[J]. 电视技术,2014,38(1):115-119.

[15] 崔宝才. 基于 GA 改进 BP 神经网络网络异常检测方法[J]. 现代电子技术,2016,39(3):90-93.

[16] 马福祥,马秀娟. 一种基于二次变异策略的改进型遗传算法[J]. 计算机工程与应用,2014,50(13):62-65.

[17] SRINIVAS M, PATNAIK L M. Adaptive probabilities of crossover and mutation in genetic algorithms[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1994, 24(4):656-667.