# Netfilter性能动态改善方法的研究与实现

杨明明,王 铮
(重庆大学 计算机学院 计算机软件与理论系,重庆 400030)

**摘 要**:Linux作为一个广泛使用且开放源码的操作系统,具有优异的网络性能,在其上布置防火墙有一个可靠的基石。特别是逐渐成熟和完善的Netfilter框架是一个非常优秀的防火墙架构。因此,越来越多的人选择Linux作为其防火墙开发平台。但是,随着规则的增加,性能成为Netfilter的一个瓶颈。提出了一种类似"冒泡算法"的算法及其实现,它能够有效改善Netfilter防火墙性能。这种算法能通过防火墙实际运行的环境自适应的动态改变Netfilter的规则匹配顺序和钩子函数调用顺序,很大程度上提高了防火墙包过滤的性能。这种算法的时间复杂度是$O(1)$。

**关键词**:Linux;Netfilter;算法;性能优化
**中图分类号**:TP309 **文献标识码**:A **文章编号**:1673-629X(2010)04-0163-04

# Study and Implementation of Algorithm to Dynamic Improvement of Performance of Netfilter

YANG Ming-ming, WANG Zheng
(Dept. of Computer Software and Thoery, Institute of Computer, Chongqing University, Chongqing 400030, China)

**Abstract**: Linux as a widely used open source operating system, has excellent network performance and reliable firewall architecture. More and more users choose Linux as a firewall platform. However, as the rules increases, the performance became a bottleneck in Netfilter. In this paper, an algorithm similar bubble algorithm is proposed and implementned which can effectively improve the performance of Netfilter firewall. This algorithm can dynamically change the order of rules and the order of hook function call for Netfilter, which largely improve the performance of packet filtering of Netfilter. And this algorithm's time complexity is $O(1)$.

**Key words**: linux; netfilter; algorithm; optimization

## 0 引 言

随着网络带宽的不断增加,网络流量的急剧飙升,防火墙的性能越来越受到关注。Netfilter作为一个优秀的防火墙系统,越来越多地被公司企业等用作网关防火墙。随着公司规模增大,各种网络应用的不断的丰富,Netfilter所使用的匹配规则和人们为实现特殊要求而增加的自定义钩子函数越来越多。此时Netfilter防火墙的性能会随之下降。针对这种情况,通过分析Netfilter的源码实现,提出一种类似于冒泡算法的方法,使得防火墙过滤包时能根据实际应用环境来自适应地动态让匹配次数多的规则和返回非NF_ACCEPTDE钩子函数向上浮,使得规则匹配的查找次数和钩子函数被"无效"调用的次数降低,从而提高了防火墙的效率。

**收稿日期**:2009-07-30;**修回日期**:2009-10-05
**作者简介**:杨明明(1984-),男,硕士研究生,主要研究方向为网络安全;王 铮,硕士,副教授,主要研究方向为网络安全、嵌入式实时操作系统。

## 1 Netfilter的框架及实现分析

### 1.1 规则添加分析

规则是通过配套工具iptables来添加。它设置防火墙的过滤规则,并将规则添加到内核空间的特定信息包过滤表内的链中,通过Netfilter框架的钩子函数完成对数据包的过滤工作。通过iptables添加的规则是顺序存储的,是一种链表结构,每个规则由ipt_entry,ipt_entry_matchs,ipt_entry_target三部分组成。其中ipt_entry结构体保存了主要的标准匹配内容和与遍历规则相关的变量target_offset与next_offset,通过target_offset可以找到规则中动作部分ipt_entry_target的位置,通过next_offset可以找到下一条规则的位置[1]。

### 1.2 Netfilter框架分析

Netfilter内核模块是完成对数据包做分析和处理的地方,它是一种内核中用于扩展各种网络安全服务的结构化的底层框架[2]。Netfilter在协议栈的包游历的路径上设置了五个钩子点(hooks)。Netfilter在每个

钩子点上为每个协议定义了不同的 hook 函数。也可以根据具体的需要在钩子点上注册自己的 hook 函数，它们组成了一个函数链。如图 1 所示。

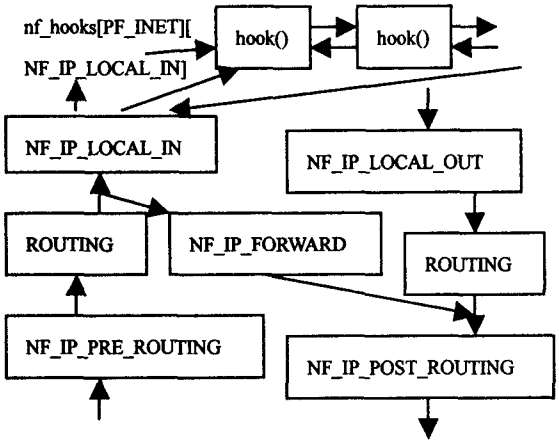


图 1 数据包经过的路线和 hook 点

正如图 1 中显示的，Netfilter 定义的五个 hook 点分别是[3]：

| Hook | 调用时机 |
|---|---|
| NF_IP_PRE_ROUTING | 在完整性校验之后，选路确定之前 |
| NF_IP_LOCAL_IN | 在选路确定之后，且数据包的目的地址是本机地址 |
| NF_IP_FORWARD | 目的地址是其他主机的数据包 |
| NF_IP_LOCAL_OUT | 来自本机进程的数据包在其离开本机的过程中 |
| NF_IP_POST_ROUTING | 数据包在准备“上线” |

Netfilter 的 hook 注册点函数指针存放在 nf_hooks 中，nf_hooks 是结构体 struct list_head nf_hooks[NPROTO][NF_MAX_HOOKS]的二维数组，其中 NPROTO 是协议族编号，NF_MAX_HOOKS 是 hook 点编号[4]。如图 1 中 nf_hooks[PF_INET][NF_IP_LOCAL_IN]，表示 IPV4 协议的 NF_IP_LOCAL_IN 钩子点上注册的 hook 函数链头，其他钩子点的情况也是一样，在此没在图中画出。接下来详细讨论这个二维数组以及 Netfilter 的规则匹配和 hook 函数的调用方式。

### 1.3 Netfilter 的规则匹配和包处理过程分析

正如前面所述，钩子函数由全局的二维链表数组 nf_hooks 保存，其按照协议族归类存储，在每个协议族中，根据钩子点的顺序排列，在钩子点内则根据钩子函数的优先级排列。钩子函数的优先级是一个整数，可以是正也可以是负。在相应的钩子点调用钩子函数时，根据协议族和钩子点找到相应的链表入口，即相应的 nf_hooks[NPROTO][NF_MAX_HOOKS]，然后依次调用该链中的每个钩子函数对数据包进行处理。

在此以 Netfilter 的 NF_IP_LOCAL_IN 钩子为例子来分析当数据包经过钩子点时 Netfilter 怎样进行规则匹配和数据包的处理(其他钩子点的处理都和 NF_IP_LOCAL_IN 是一样的过程)，如图 2 所示。

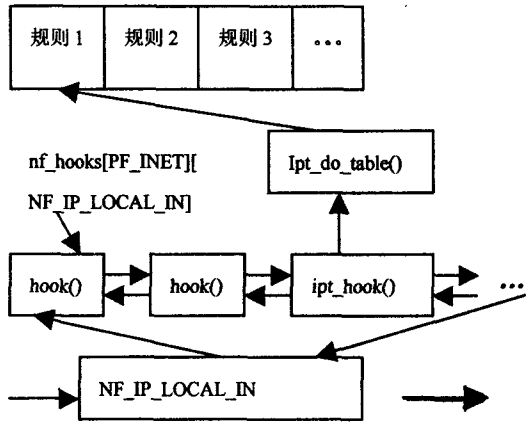


图 2 规则匹配和包处理过程

当一个 TCP/IP 数据包到达 NF_IP_LOCAL_IN 钩子时，Netfilter 通过包的协议族和 NF_IP_LOCAL_IN 就可以确定 nf_hooks[PF_INET][NF_IP_LOCAL_IN]，nf_hooks[PF_INET][NF_IP_LOCAL_IN]指向了钩子函数链，如图 2 的 hook 函数链表，函数是按照优先级升序排列的。Netfilter 中默认表 filter 在建立时在 NF_IP_LOCAL_IN 钩子点注册了函数 ipt_hook()，ipt_hook()会调用 ipt_do_table()函数，这个函数会对相应的表和钩子点的规则进行遍历处理。而其他的 hook 函数则是用其他的方式加载上去的，比如自己为满足实际情况中特殊要求编写的 hook 函数模块等。找到 nf_hooks[PF_INET][NF_IP_LOCAL_IN]后，Netfilter 会从第一个钩子函数起，依次调用接下来的钩子函数。hook 函数的返回值只能是以下几种情况[5]：

| 返回值 | 含义 |
|---|---|
| NF_DROP | 丢弃数据包 |
| NF_ACCEPT | 接收数据包 |
| NF_STOLEN | 模块接管该数据包，不要再继续传输该数据包 |
| NF_QUEUE | 对该数据包进行排队(通常用于将数据包传给用户空间的程序进行处理) |
| NF_REPEAT | 重复调用该 hook 函数 |

hook 函数链表处理的核心函数是 nf_iterate 函数。这个函数通过一个 for 循环遍历所有的 hook 函数。它有多种返回值，并且对不同返回值做不同的处理。如果返回值是 NF_QUEUE，NF_STOLEN，NF_FROP，就立即返回，不再调用后面的 hook 函数，如果返回值是 NF_REPEAT，只是改变一下当前循环中的指针，然后将从新调用这个 hook 函数。然而只有所有这个链表上的 hook 函数的返回值都是 NF_ACCEPT

时,这个函数才会返回 NF_ACCEPT,此时包才会继续在 TCP/IP 栈中游历[6]。

对于规则匹配,上面已有提及,它是 Netfilter 默认注册的钩子函数 ipt_hook()调用 ipt_do_table()函数处理的,我们关心的匹配处理过程则是从规则链头开始依次匹配,直到遇到匹配的规则或链表结尾才返回。

### 1.4 问题的提出与解决思路

通过上两段的分析可以发现,不管是 hook 函数的调用还是规则的匹配,都是一个链表的顺序查找问题,这是影响 Netfilter 包过滤效率的关键问题之一。随着网络流量的激增和网络应用的丰富,规则表链和钩子函数链的规模会不断增大,这样 Netfilter 的包过滤效率会明显降低。这个问题在 Netfilter 中并没有被很好解决。对于规则表链中的规则,它们只是按人们用 iptables 输入的先后顺序在链表中排列,这样可能会使得经常被匹配的规则在后面,而很少用到的规则却在前面,使得匹配过程做很多无用功,降低了匹配效率。对于钩子函数的调用,它们虽然是按人为设定或系统默认的优先级排序了,但是优先级的设定是在编写 hook 函数模块的时候在程序中设定,以后也不能根据使用中的具体实际情况改变。同时注意到,不管是 hook 函数的调用还是规则的匹配,都有立即返回的条件,那就是规则被匹配和 hook 函数返回值不是 NF_ACCEPT。并且,通过收集分析现实中防火墙规则匹配和钩子函数的返回值的统计数据,发现一定环境下规则匹配的次数和 hook 函数返回非 NF_ACCEPT 的次数都呈较稳定正态分布,但是不同环境分布不一样。对于规则匹配,如果有一种算法能使得被匹配次数或者说频率高的规则更靠前的话,规则匹配就会减少匹配次数(也就是链表查找的比对次数减少),从而规则匹配的效率会明显的提高。同样,对于钩子函数链,如果能够使得处理数据包后的返回值不是 NF_ACCEPT 的次数多或频率高的 hook 函数更靠前,那么整个函数立即返回的机会就明显增加,同样提高了效率。

## 2 使用类冒泡算法优化 Netfilter 效率

要解决上面的问题,需要一种有效的能够根据防火墙所在环境而自动地调整规则和 hook 函数的顺序的方法。我们设计一种类似冒泡原理的算法,使得被匹配次数多的规则和返回值不是 NF_ACCEPT 的次数多 hook 函数会自动"向上浮",从而达到提高效率的作用。设计的算法是:对于规则,当一条规则被匹配的时候,此规则就与它的前一条规则交换位置,这样匹配次数越多的规则就会越靠前。而前面提到了,在一定环境下规则匹配的次数和 hook 函数返回非 NF_ACCEPT 的次数都呈较稳定的正态分布,这样防火墙就可以通过对环境的适应达到一个稳定而高效的状态。对于 hook 函数也是一样,当返回非 NF_ACCEPT 时,与前一个 hook 交换位置,只是这里比规则那要多一步,就是要计算出 hook 函数的新优先级,使得新的优先级在其上下两钩子函数中间以保持钩子函数链的原有排序,以使得不对原内核程序造成影响。

因为规则和 hook 函数都是以双向链表的形式存放,所以,我们的算法只需要改变几个指针,至于计算新的优先级,也只需用上下 hook 的优先级求中间值就行了。所以算法的时间复杂度为 $O(1)$。因此即使最坏的情况出现,也不会对原程序造成性能的影响,一般情况则会较大程度地提高 Netfilter 的效率。

## 3 类冒泡算法的实现与测试

### 3.1 算法的实现

算法实现并不复杂,只要找到 Netfilter 中相应的处理函数,并在 return 前将当前规则或 hook 函数的位置前移一位就行了。对于规则表链,Netfilter 是调用 ipt_do_table 函数来处理的。这个函数很长,但是它只有两个 return 的地方,实现代码和在 hook 处理程序中的片段的"前移一个位置"注释下面的代码实现了相同的功能,这里就不列出来。对于 hook 函数链,这里先介绍结构体 nf_hook_ops[7]:

```
struct nf_hook_ops{struct list_hesd list;
nf_hookfn * hook; //钩子函数
struct module * owner; //拥有者
int pf; //协议族
int hooknum; //hook 点
int priority;}; //优先级
```

前面提到的钩子函数链其实每个节点都是一个 nf_hook_ops 结构,其中的 hook 成员就是钩子函数,priority 则是要设定和改变的 hook 函数优先级。而 pf 和 hooknum 则是 nf_hooks[][]的下标,帮助确定 hook 函数在哪个链上。Netfilter 对 hook 函数的调用是在 nf_iterate 中,这里把需要改变的部分截取出来[8]:

```
static unsigned int nf_iterate(.....){
for(.....)
switch(.....){
case NF_QUEUE:
return NF_QUEUE;
case NF_STOLEN:
    return NF_STOLEN;
case NF_FROP:
    return NF_DROP;
⋮
```

```
}}
```

只要在三个 return 前加上这段代码：

```
struct list_head *elem1 = (struct list_head *)elem //elem为当前节点
if(elem1->prev != head){
//前移一个位置
elem1->prev->next = elem1->next;
elem1->next->prev = elem1->prev;
elem1->next = elem1->prev;
elem1->prev = elem1->next->prev;
elem1->prev->next = elem1;
elem1->next->prev = elem1;
//重新设置优先级,保持与原程序的一致性,使得修改不会引起原程序的变化
struct nf_hook_ops *elemprev = (struct nf_hook_ops *)elem1->prev;
struct nf_hook_ops *elemnext = (struct nf_hook_ops *)elem1->next;
(elemprev->priority%2==1&& elemnext->priority%2==1)? elem->priority=(elemprev->priority/2 + elemnext->priority/2 +1):(elemprev->priority/2 + elemnext->priority/2);
}
```

### 3.2 算法测试

本实验的测试环境是在实验室自行搭建的局域网上，在自然使用的条件下测试的。防火墙主机采用的操作系统是 centos5.2，因为内核模块编程的需要，所以在其上下载了版本是 2.6.18 的内核源代码自己编译安装了一个系统，试验就是在 linux-2.6.18 内核上进行的。CPU 是 Intel Celeron 1.60G，内存 0.99G，自适应 10/100M 网卡双网卡。在 filter 模块的钩子点 NF_IP_FORWARD 设置了 200 条规则，并在上面加载了 20 个实现单一功能的 hook 函数模块。还编写了一个统计模块 statist.ko，用来统计每个 hook 函数返回非 NF_ACCEPT 的次数和每条规则匹配的次数。通过一段时间的自然使用，对比了 hook 函数返回非 NF_ACCEPT 的次数和 hook 函数在 hook 函数链表中的位置，发现返回非 NF_ACCEPT 的次数越多，相应 hook 函数的位置越靠前。同样规则的匹配次数越多，位置越靠前。与预期一样，证明算法是有效的。

## 4 结束语

Netfilter 被认为是一个非常强大的内核防火墙系统。但是由于其规则匹配和 hook 函数调用都采用线性的方式，当规则集很大，挂载的钩子函数较多时，其性能会有明显的降低。我们的算法通过让防火墙根据环境自适应的动态调整规则和 hook 函数的次序，减少了规则匹配的次数和 hook 函数“无效”调用的次数，从而提高了防火墙的效率。当然，提高 Netfilter 效率的还有其他的方法，比如防火墙的体系结构，防火墙的负载平衡等。

**参考文献：**


[1] 倪继利. linux 安全体系分析与编程[M]. 北京：电子工业出版社，2007.

[2] 朱立才，杨寿保，宋舜宏. Netfilter/iptables 防火墙性能优化方案与实现[J]. 计算机工程与应用，2006，42(15)：117-120.

[3] Baba T，Matsuda S. Tracing Network Attacks to Their Sources[J]. IEEE Internet Computing，2002，62(2)：107-109.

[4] 毛德操，胡希明. Linux 内核源代码情景分析[M]. 杭州：浙江大学出版社，2006.

[5] 博嘉科技. linux 防火墙技术探秘[M]. 北京：国防工业出版社，2002.

[6] 王丽辉，李 涛，张晓平，等. 一种联动防火墙的网络入侵检测系统[J]. 计算机应用研究，2006，23(3)：95-97.

[7] Bllovin S M. Secruity problems in the TCP IP protocol suite [J]. Computer Communications Review，1998，92(2)：81-83.

[8] Steven M B，William R C. Network Firewalls[J]. IEEE Communications，1994，9(9)：67-69.


(上接第 162 页)


ty Foundations Workshop. Washington，DC，USA：IEEE Computer Society，1997：109-115.

[5] Lye Kong-wei，Jeannette M W. Game strategies in network security[J]. International Journal of Information Security，2005，4(1-2)：71-86.

[6] 孙 薇，孔祥维，何德全，等. 基于演化博弈论的信息安全攻防问题研究[J]. 情报科学，2008(9)：1408-1412.

[7] 朱建明，Raghunathan S. 基于博弈论的信息安全技术评价模型[J]. 计算机学报，2009(4)：828-834.

[8] 贾春福，钟安鸣，张 炜，等. 网络安全不完全信息动态博弈模型[J]. 计算机研究与发展，2006，43(z2)：530-533.

[9] 谢识予. 经济博弈论[M]. 第 2 版. 上海：复旦大学出版社，2002：20-41.

[10] 曹 晖，王青青，马义忠. 基于动态贝叶斯博弈的攻击预测模型[J]. 计算机应用，2007(6)：1545-1547.

[11] 何 宁，卢 昱，王 磊. 网络控制论在网络攻防中的应用[J]. 武汉大学学报：理学版，2006(5)：639-643.