第35卷 第11期 计 算 机 工 程 2009年6月
Vol.35 No.11 Computer Engineering June 2009
·安全技术· 文章编号：1000—3428(2009)11—0143—03 文献标识码：A 中图分类号：TP301.6

Linux下基于Netfilter的包过滤算法

刘 云
(贵阳学院数学系，贵阳 550003)

摘 要：通过对Linux操作系统下Netfilter防火墙中包过滤技术的分析，发现Netfilter包过滤使用简单的线性分级算法，当防火墙需要匹配的规则越来越多时，防火墙的性能会急剧下降，造成系统瓶颈。因此，提出一种基于二叉树和Hash函数的包过滤算法B-H。通过测试证明，该算法在大量规则的情况下能够达到快速匹配，有效地提高了包过滤的性能。

关键词：Linux操作系统；Netfilter防火墙；包过滤；二叉树；Hash函数

Packet Filtering Algorithm Based on Netfilter Under Linux

LIU Yun
(Dept. of Mathematic, Guiyang University, Guiyang 550003)

【Abstract】This paper analyzes the packet filtering technique based on Linux operating system. The packet filtering of Netfilter uses simple linearity graduation algorithm. When the firewall needs to match the number of rules, the performance of firewall falls suddenly, and it becomes the system bottleneck. Therefore, a new packet filtering algorithm based on binary tree and Hash function is proposed, that is B-H. Test proves that the algorithm can achieve the fast match in the massive rules, and enhance the performance of packet filtering greatly.

【Key words】Linux operating system; Netfilter firewall; packet filtering; binary tree; Hash function

1 概述

在信息化社会中，网络信息系统在政治、军事、金融、商业、交通、电信、文教等方面发挥越来越大的作用。随着网络规模的不断扩大与应用技术的不断进步，越来越多的业务需要对数据包进行实时、快速的分类过滤。数据包过滤是一个用软件或硬件设备对向网络上传或从网络下载的数据流进行有选择的控制过程。数据包过滤技术是防火墙的基本技术，它对IP数据包的包头进行检查以确定数据包的源地址、目的地址和数据包利用的网络传输服务。

在因特网的分层模型中，要传输的数据被各层协议的包头依次封装着，每一层的包头都包含若干域，它们分别携带着该层协议的特征数据。包过滤规则rule可以涉及到从数据链路层到应用层的任何域，它对所涉及的域及域的取值或取值范围加以定义，若干个涉及相同域的包过滤规则的集合构成包过滤集[1]。

2 Linux下的Netfilter框架结构

Linux操作系统是一个支持多用户、多任务、多进程、实时性较好、功能强大而稳定的操作系统，也是目前运行硬件平台最广泛的操作系统。Linux的防火墙技术经历了若干代的改革，一步步地发展而来。Linux下Netfilter/Iptables应用程序被认为是Linux中实现包过滤功能的第4代应用程序。Netfilter/Iptables包含在2.4以后的内核中，它可以实现防火墙、数据包过滤、网络地址翻译(NAT)和数据包分割等功能。Netfilter工作在内核，而Iptables是让用户定义规则集的表结构，可以用Iptables为Unix, Linux和BSD个人工作站创建一个防火墙，也可以为一个子系统创建防火墙以保护其他的系统平台。

Netfilter作为数据包分割的框架，不属于普通的Berkeley套接字接口。首先，每个协议定义了“挂钩(hooks)”，用来定义遍历那个协议栈的指针。用一个二维数组结构存储，一维为协议族，二维为上面提到的各个调用入口。简单地说，Netfilter的架构就是在整个网络流程的若干位置放置了一些检测点(HOOK)，如图1所示。而在每个检测点上登记了一些处理函数进行处理(如包过滤、NAT等，甚至可以是用户自定义的功能)。

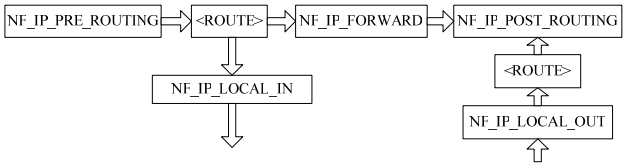

图1 Netfilter HOOK的位置

其次，对于每一个协议来说，内核部分能常驻监听不同的挂钩，所以当一个包经过Netfilter框架时，它会被检查是否已经驻在了那个协议和挂钩。如果是，它们每一个都有一个机会按照顺序检查一遍包，然后丢弃包(NF_DROP)或允许包经过(NF_ACCEPT)，或告诉Netfilter忘记包(NF_STOLEN)，或让Netfilter为用户空间排列一下包(NF_QUEUE)。第3部分是那些被选择了排列后送往用户空间的包(按照ip_queue驱动)；这些包被异步处理[2]。

3 Netfilter功能的实现

Netfilter/Iptables由2部分组成，一部分是Netfilter的钩子函数“hooks”，另一部分则是知道这些钩子函数如何工作

作者简介：刘 云(1981－)，女，硕士研究生，主研方向：信息安全，网络应用

收稿日期：2008-11-03 **E-mail：**cloudgsy@163.com

的一套规则，这些规则存储在 Iptables 数据结构之中。钩子函数通过访问 Iptables 来判断应该返回什么值给 Netfilter 框架。在 Iptables 中一个表(table)就是一组类似的防火墙 rules 的集合。Iptables 里面默认定义了 3 个 table：filter, mangle 和 nat。

(1)filter 表。该表不会对数据包进行修改，而只对数据包进行过滤。它通过钩子函数 NF_IP_LOCAL_IN, NF_IP_FORWARD 及 NF_IP_LOCAL_OUT 接入 Netfilter 框架。因此，对于任何一个数据包只有一个地方对其进行过滤。

(2)nat 表。监听 3 个 Netfilter 钩子函数：NF_IP_PRE_ROUTING, NF_IP_POST_ROUTING 及 NF_IP_LOCAL_OUT。NF_IP_PRE_ROUTING 实现对需要转发的数据包的源地址进行地址转换而 NF_IP_POST_ROUTING 则对需要转发的数据包的目的地址进行地址转换。对于本地数据包的目的地址的转换则由 NF_IP_LOCAL_OUT 来实现。

(3)mangle 表。在 NF_IP_PRE_ROUTING 和 NF_IP_LOCAL_OUT 钩子中进行注册。使用 mangle 表可以实现对数据包的修改或给数据包附上一些带外数据。

Iptables 防火墙规则的组织结构中第 1 级是 table 级，每一个防火墙可以有多个 table；第 2 级是 hook 级，每一个 table 都有一个 hook 集合，每个 hook 都有一个防火墙规则链；第 3 级是基本规则级。基本规则级的规则包括 3 部分：IP 规则信息，匹配规则信息和 target。而这 3 个组成部分的每一个都可以包括同类型的多个部分规则。

table, chain 和 rule 三者的关系如下：table 是实现某项功能所有规则的总和；chain 是在某个检查点上所引用规则的集合；rule 是一个单独的规则。match 在规则中用于匹配数据包中的各项参数；target 在规则中决定如何处理匹配到的数据包，因此，在 target 中实现了具体的网络安全功能[3]。

4 Netfilter数据包过滤的实现

数据包由左边进入，先经过了简单的检测(如没有被截断、P 校验和正确、无混杂随意接受)，然后被传送给网络过滤框架的 NF_IP_PRE_ROUTING 挂钩。接着，它们进入路由选择阶段，这个阶段将决定包是被送到另一个接口，或者属于一个本地的进程。这个阶段可能会抛弃不能进行路由选择的包。如果包是属于本地的，那么在包被送给进程前，网络过滤框架的 NF_IP_LOCAL_IN 挂钩将被使用。如果包被送到另一个接口，那么网络过滤框架的 NF_IP_FORWARD 挂钩将被使用。最后，在包再次进入网络之前，会被传送到 NF_IP_POST_ROUTING 挂钩。NF_IP_LOCAL_OUT 挂钩将被属于本地的包使用。

NF_IP_PRE_ROUTING：刚刚进入网络层的数据包通过此点(刚刚进行完版本号、校验和等检测)，源地址转换在此点进行。

NF_IP_LOCAL_IN：经路由查找后，送往本机的通过此检查点，INPUT 包过滤在此点进行。

NF_IP_FORWARD：要转发的包通过此检测点，FORWORD 包过滤在此点进行。

NF_IP_POST_ROUTING:所有马上便要通过网络设备出去的包通过此检测点，内置的目的地址转换功能(包括地址伪装)在此点进行。

NF_IP_LOCAL_OUT:本机进程发出的包通过此检测点，OUTPUT 包过滤在此点进行。

Netfilter 数据包过滤是在 filter 表中实现的，该模块的功能是过滤报文，不作任何修改，或者接受，或者拒绝。它在 NF_IP_LOCAL_IN, NF_IP_FORWARD 和 NF_IP_LOCAL_OUT 3 处注册了钩子函数，也就是说，所有报文都将经过 filter 模块的处理。当数据包到达防火墙时，内核先检查数据包的头信息，根据情况将数据包送往包过滤表(filter)的不同的链(chain)。将信息包的头信息与它所传递到的链中的每条规则进行比较，看它是否与某条规则完全匹配。

(1)如果数据包与某条规则匹配，那么内核就对该信息包执行由该项规则的目标指定的操作。如果目标为 ACCEPT，则允许该信息包通过，并将该包发给相应的本地进程处理。如果目标为 DROP 或 REJECT，则不允许该信息包通过，并将该包阻塞并杀死。

(2)如果数据包与这条规则不匹配，那么它将与链中的下一条规则进行比较。

(3)如果数据包与链中的任何规则都不匹配，那么内核将参考该链的策略来决定如何处理该数据包。理想的策略应该告诉内核 DROP 该数据包。

在 Netfilter 中数据包过滤采用 Hash 索引与简单线性链表相结合的方法。在建立规则库时根据每一条规则的源地址、目的地址建立 Hash 索引，接着将与之对应的规则存入该条规则 Hash 索引所指向的存储单元。所以对规则库中规则的添加、插入、删除就是对 Hash 索引和与之对应的线性链表进行添加、插入、删除操作[4]。

当一个数据包进入 Netfilter 的包过滤模块时，它的匹配步骤如下：

(1)根据该数据包的源地址、目的地址进行 Hash 计算，计算出关键字 k。Hash 函数可选取折叠法加除留余数法。

(2)根据 Hash 计算的结果关键字的值，到 Hash 索引表中进行查找。如果找到则进入与之对应的规则线性链表中进行查找匹配。如果找不到，则转到(3)。

(3)如果找不到则说明这是一个新流，为这个流建立一个新的 Hash 表项，插入到关键字 k 对应的同义词子表中。

5 Netfilter包过滤算法的改进

一个数据包进入 Netfilter 后，首先组成碎片(将包头中所需的数据提出来)，组完后查询一遍当前的连接表(第 1 个表，Hash 链表结构)，虽然是 Hash 结构，但是当数量变大时每个链表里的索引信息还是很多。如果 Hash 链表中查不到，那么先分配内存建立这个数据包的连接信息(也就是创建该数据包的 Hash 值插入 Hash 链表中)，然后查询期待子连接列表(第 2 个表，链表)，这个表是单一线性表。如果在链表中查找到该记录，则将 Hash 链表中的索引与线性链表中找到的记录绑定在一起。最后根据各个 IP 协议的处理检查数据包的合法性，这个在用户层不可控，由 Netfilter 自动完成。接着，数据包进入 PREROUTING 点查询目的 NAT 表(第 3 个表，数组结构)，若匹配则进行目的 NAT，否则进行一个空绑定，每个后续包都要到这个绑定函数里操作一番。接着进行路由表查找，判断该数据包或者转发，或者进入本机。如果转发，需要查转发链的规则表(第 4 个表，数组结构)进行过滤。最后数据包进入 POSTROUTING 点，和 PREROUTING 的目的 NAT 类似，但这里是进行源 NAT，同样要查源 NAT 规则表(第 5 个表，数组结构)。如果匹配这里的规则则按规则转换，若这里没有规则匹配该数据包，则进行地址不变的转换。

由此可见，一个包进了 Netfilter 至少要查找 5 个表。其中，连接状态表是数量最大的，所以也是用 Hash 形式处理。

其次应该属于转发链的过滤表，如果不把状态检查规则放前面，则每个后续包检查的规则也不会少，尤其是作流量控制时。另外，前后 2 次 NAT 绑定操作也是特别费性能的操作，而且如果本身不进行 NAT，也得进行空绑定操作，空绑定时所有流程也走一遍，并没有进行简化，校验和都要计算一遍。

由于 Linux 下的 Netfilter/Iptables 中包过滤使用简单的线性分级算法，这样缺乏效率，特别是当网络变得越来越复杂、防火墙需要匹配的规则越来越多时，防火墙的性能将会急剧下降，造成系统瓶颈。因此提出了基于二叉树和 Hash 函数的包过滤算法——B-H，大大提高了包过滤的性能。

二叉树结构是一维树结构的直接扩展：如果维数 $d>1$，那么先建立第一维的树结构。从根节点开始，根据规则的由高到低的关键字值建立树的每一条分支(1 对应右分支，0 对应左分支)。如果没有下一个域搜索，则结束节点为终节点，该节点存有对应的规则号，否则该节点仍为中间节点，节点中的下一条指针指向一个域的根节点。

5.1 二叉树算法的构成

二叉树算法由 2 部分构成：生成树结构部分和匹配部分。二叉树结构的构造是算法的基础部分，每一个条件域都要构造一棵二叉树结构；匹配算法先在所有的二叉树结构中搜索可能的匹配规则，然后用二分法查找最终的结果。

5.2 二叉树结构的构造

二叉树结构的构造方法是根据二进制值确定分支方向，0 为左分支，1 为右分支。叶节点存储规则为：对于规则库中某条规则的某一个条件域，若其前缀表达式与二叉树结构的某条分支匹配，则该分支的叶节点记录该规则，叶节点存储的是一个有序队列。

对一个分组查找匹配的规则从根节点开始，根据分组中的数据是 0 还是 1 来选择左子树或右子树，查找过程中保存在路径上最近遇到的一条规则。当最后到达一个空节点时，最近的那条规则就是该分组匹配的规则。因为前缀限制短的规则必然比前缀限制长的规则先被遇到，所以最后遇到的规则就是最长前缀匹配的规则。

5.3 匹配算法

构成规则树后，就可以对到来的数据包进行匹配。最简单的就是二分法，在一维空间上进行等子空间的划分，由于查找次数和总的数据量呈指数关系，因此二分法查找在大多数情况下优于直接对数据集线性查找。

B-H 算法只用到 Netfilter 里 5 个钩子 HOOK 中的 3 个：NF_IP_LOCAL_IN, NF_IP_FORWARD 和 NF_IP_LOCAL_OUT，用来进行数据包过滤处理，如图 2 所示。

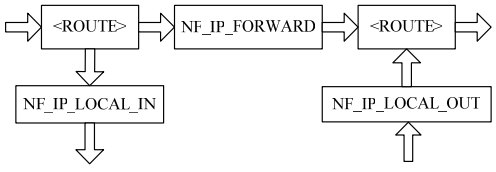

图 2 B-H 算法使用的 HOOK

一般情况下，过滤规则基本上只限于 5 个域(源/目的地址、源/目的端口、协议域)和 1 个动作(接受或拒绝)。将这 5 个域连接成比特串，共 104 bit，接下来建立这 104 bit 信息的 Hash 映射，如图 3 所示。

一个数据包进入 B-H 后，将数据包的包头中源/目的地址、源/目的端口、协议组成一个 5 元组。然后计算出该数据包的 Hash 关键字 k，将 k 与当前的连接表的 Hash 索引中的关键字进行比较(采用二分法查找)。如果在 Hash 链表中查不到，那么就得先分配内存建立这个数据包的连接信息(也就是创建该数据包的 Hash 关键字 k，插入 Hash 索引二叉树中)。如果找到与该数据包对应的规则，则按照规则的动作进行接受/拒绝操作；如果找不到对应的规则，则按默认操作处理该数据包[5]。

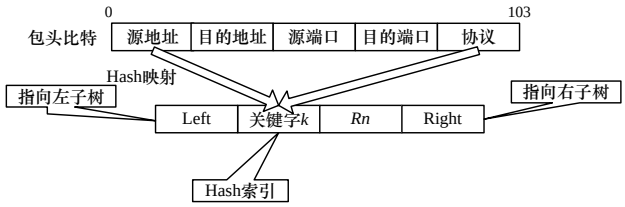

图 3 数据包包头

6 性能测试

用 Netperf 软件测试 Netfilter 和 B-H 算法对数据包过滤的性能，该软件主要针对基于 TCP 或 UDP 的传输。Netperf 测试结果反映的是一个系统能够以多快的速度向另外一个系统发送数据，以及另外一个系统能够以多快的速度接收数据。

将 2 台计算机用直连双绞线连接，一台计算机产生 TCP 数据包并测试吞吐量，另一台计算机上分别运行 Netfilter 和 B-H 2 套程序。结果显示出在规则数很大的情况下，B-H 比 Netfilter 的性能要高得多。随着规则数的增加，Netfilter 性能下降很快。

7 结束语

通过对 B-H 中的包过滤算法的分析，发现 B-H 使用一个简单的包分类算法，对线性地穿过一个链中的每一个包顺序匹配一条一条的规则。所以它的算法是使用线性链表来匹配规则，这个方法非常缺乏效率。线性链表在规则数目很少的情况下比较合适，但当规则数目很大时，性能也随之下降，性能与规则数是线性关系。

在了解该算法后对 B-H 的包过滤进行改进，提供一个新颖的包过滤架构。当查找每一个包的时候使用一个高级算法来减少内存占用，充分优化以实现适度内存占用和高性能的包分类完全动态。当插入或者删除规则时数据结构没有重建，高速更新成为可能。因此，提出基于二叉树和 Hash 函数的包过滤算法 B-H，理论上二叉树算法比线性链表的效率高很多。最后的测试也证明了 B-H 包过滤算法在大量规则的情况下能够达到一种快速匹配。网络性能与规则数也尽可能无关，无论多少规则，匹配速度基本上是相同的，它与硬件实现的目标基本一致。

参考文献

[1] Wang Dong, Hao Ruibing, Lee D. Fault Detection in Rule-based Software Systems[J]. Information and Software Technology, 2003, 45(12): 865-871.

[2] 刘建峰，潘 军，李祥和. Linux防火墙内核中Netfilter和Iptables的分析[J]. 微计算机信息, 2006, 22(3): 7-8.

[3] 王永滨. Linux 防火墙规则的可视化输入与翻译[J]. 计算机应用研究, 2001, 18(12): 107-108.

[4] 田大新，刘衍珩，李永丽，等. 数据包过滤规则的快速匹配算法和冲突检测[J]. 计算机研究与发展, 2005, 42(7): 1128-1135.

编辑 顾逸斐