openwrt linux 基于iptables的流量和速度统计

1. **简介**

    目前openwrt系统中流量统计做的最好的应该是“石像鬼”固件了，用以做流量统计的工具也有很多如：tomato，luci-app-statistics等。

    本文想给大家介绍一种基于iptables规则的流量统计方法。该方法的基本原理是利用iptables自带的对规则链的流量统计功能，通过制定不同的规则并挂在不同的表和链上来实现对特定流量统计。

    本方法的好处如下：

* + 灵活统计多种流量。
  + 对系统性能几乎不影响。
  + 流量统计准确。
  + 便于扩张。

**iptables简介**

**iptables表说明**

**mangle 表**   #主要作用是更具规则修改数据包的标志，以便其他规则或应用程序对其进行                 处理。做好不要在该表中做DROP处理。

**-->PREROUTING**   #在执行路由决策前的数据经过该链。

**-->INPUT**      #本机接收的数据包经过该链。

**-->FORWARD**     #需要转发出去的数据包经过该链。

**-->OUTPUT**    #本机发出的数据包经过该链。

**-->POSTROUTING**  #在执行完路由决策后即将发送出去的数据包经过该链。

**nat  表**   #顾名思义该表主要做网络地址装换的。如：SNAT DNAT REDIRECT。该表不能                对数据包执行丢弃动作。

**-->PREROUTING**   #在执行路由决策前的数据经过该链,可在该链上做REDIRECT

**-->POSTROUTING**  #在执行完路由决策后即将发送出去的数据包经过该链。SNAT

**-->OUTPUT**    #本机发出的数据包经过该链。

**filter 表**  #在该表上主要做数据包的过滤。

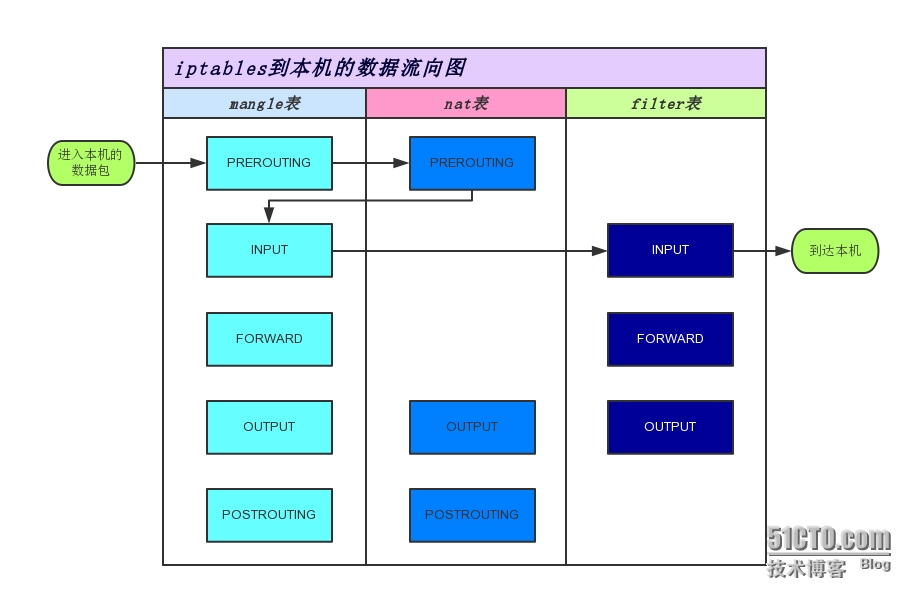
**-->INPUT**      #到本机数据包的的过滤。

**-->FORWARD**     #转发数据包的过滤。

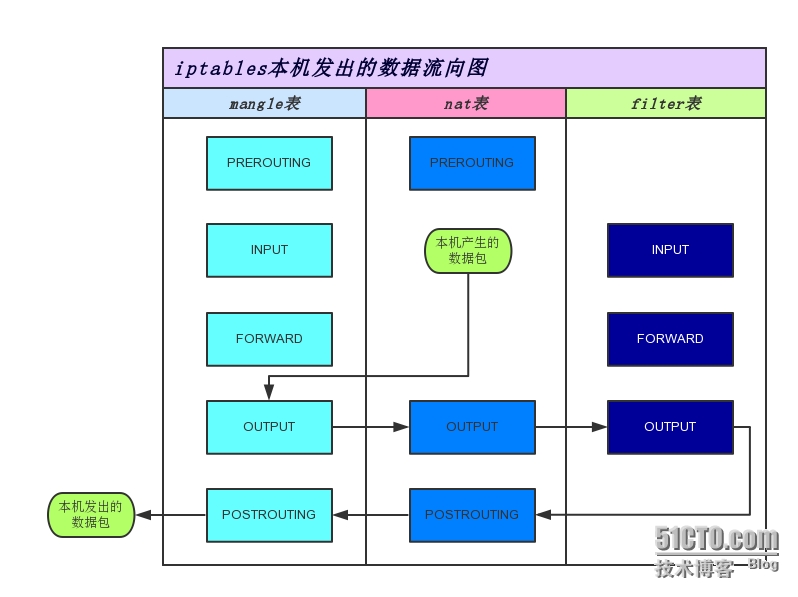
**-->OUTPUT**    #本机发出的数据包的过滤。

**iptables数据流向说明**

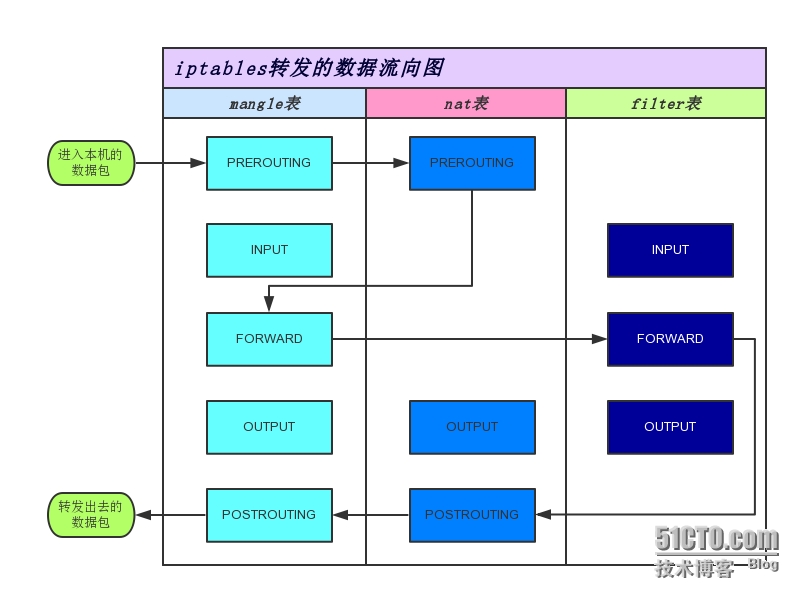
**目的地为本机的数据流向图如下：**

[](http://s3.51cto.com/wyfs02/M02/72/67/wKioL1XjJgyymYqZAAHC1sf8_xQ821.jpg)

**本机发出的数据流向图**

[](http://s3.51cto.com/wyfs02/M01/72/67/wKioL1XjJmmjN3yrAAHLHd-BtS4827.jpg)

**本机转发的数据流向图**

[](http://s3.51cto.com/wyfs02/M01/72/67/wKioL1XjJoTylDh_AAHIW70wHbM664.jpg)

**特别说明：如果已经建立连接的TCP数据是不会经过nat表的PREROUTING链的。**

**统计规则链的挂载说明**

    根据以上说明，对于统计规则链最好放在filter表中，且同时为了提高统计的精确性，需要改造filter表的上的所有ACCEPT的规则到同一自定义链上。如所有INPUT链上的ACCEPT动作都重定向到input\_accept中，OUTPUT链上的ACCEPT动作都重定向到output\_accept链上，同理FORWARD链上的所有ACCEPT规则都重定向到forward\_accept链上。

    然后分别在自定义的\*\_accept链上增加对各个客户端的统计。其中input\_accept可以统计所有客户端到本机的上行流量。 output\_accept可以统计所有客户端到本机的下行流量。 forward\_accept可以统计每个客户端经本机转发的上外网的上行与下行流量。

**代码实现说明**

    对于实现该功能主要有几个重点需要说明：

    1、如何发现接入的设备和断开的设备，从而为每个设备添加其对应的统计规则。

    2、如何高效的获取iptables规则链上的统计数据。

    3、对于及时数度的计算

**发现设备接入和断开说明**

    我的实现是每隔一秒去读取/proc/net/arp文件，通过与现有的接入链表对比，从而找到新增的断开的设备，从而对应的添加和删除其统计规则。

    这样实现的好处有：1、不会影响数据流向。2、能准确的发现设备的接入和断开事件。3、对系统性能影响不大。

    实现代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | static void handle\_client\_item(const char \*ip, const char \*mac, const char \*device)  {      struct client\_info \*client = NULL;        if (strcmp(device, LAN\_IFNAME)) {          debug(MSG\_DEBUG, "unsupport client from device:%s", device);          return ;      }        client = find\_client\_by\_ipmac(ip, mac);      if (client == NULL) {          debug(MSG\_DEBUG, "add new client ip:%s mac:%s device:%s", ip, mac, device);          client = append\_new\_client(ip, mac);      }      assert(client);        client->client\_exist = true;  }    static void handle\_arp\_line(char \*line)  {      int   count = 0;      char \*delim = " ", \*p = NULL;      char \*ip = NULL, \*mac = NULL, \*device = NULL;        p = strtok(line, delim);      do {          switch (count)          {          case 0:              ip = p;              break;          case 3:              mac = p;              break;          case 5:              device = p;              break;          }          count++;      } while ((p = strtok(NULL, delim)) != NULL);        if (check\_ip\_valid(ip) == 0) {          debug(MSG\_DEBUG, "get ip:[%s] from arp error!", ip);          return;      }      if (check\_mac\_valid(mac) == 0) {          debug(MSG\_DEBUG, "get mac:[%s] from arp error!", mac);          return;      }      if (device == NULL || strlen(device) <= 0) {          debug(MSG\_DEBUG, "get device name error!");          return;      }        handle\_client\_item(ip, mac, device);  }    static void do\_find\_new\_client()  {      FILE \*fp = NULL;      char  line[1024] = {0};        fp = fopen("/proc/net/arp", "r");      if (fp == NULL) {          return ;      }        update\_client\_no\_exist();        while (fgets(line, sizeof(line), fp))      {          if (strlen(line) <= 0) {              continue;          }          if (strstr(line, "IP") || strstr(line, "HW")) {              continue;          }          // delete the '\n'          line[strlen(line)-1] = '\0';          handle\_arp\_line(line);      }        fclose(fp);        do\_network\_stats();        delete\_unexist\_client();  } |

**IPTABLES规则的高效统计说明**

    为了实现高效的对iptables规则链上的流量统计，程序可以基于libiptc库来实现。实例代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | void do\_network\_stats()  {      struct iptc\_handle \*handle = NULL;      const struct ipt\_entry \*entry  = NULL;        handle = iptc\_init("filter");      if (!handle) {          debug(MSG\_ERROR, "iptc\_init err msg:%s", iptc\_strerror(errno));          return ;      }        entry = iptc\_first\_rule("input\_accept", handle);      while (entry) {          ip = (char \*)inet\_ntoa(entry->ip.src);          debug(MSG\_INFO, "stats ip:%s pcnt:%llu bcnt:%llu",                  ip, entry->counters.pcnt, entry->counters.bcnt);          update\_flow(ip, entry->counters.pcnt, entry->counters.bcnt);            iptc\_zero\_counter("input\_accept", count++, handle); //统计完成后清空规则链上的数据，用以实现增量统计，从而解决iptables规则被删除后重新加载导致的流量统计不正确的问题。          entry = iptc\_next\_rule(entry, handle);      }      iptc\_commit(handle); //用以应用上面清空规则链上的统计信息的代码。      iptc\_free(handle);  } |

**对于统计设备的及时速度的说明**

    当把一段事件的流量除以统计间隔，就得到对应流量的速度了。（可以一秒统计一次流量的增量，它就是1秒的平均速度了。）

**后续说明**

    如果各位想要了解更多的实现细节可以在评论在探讨。