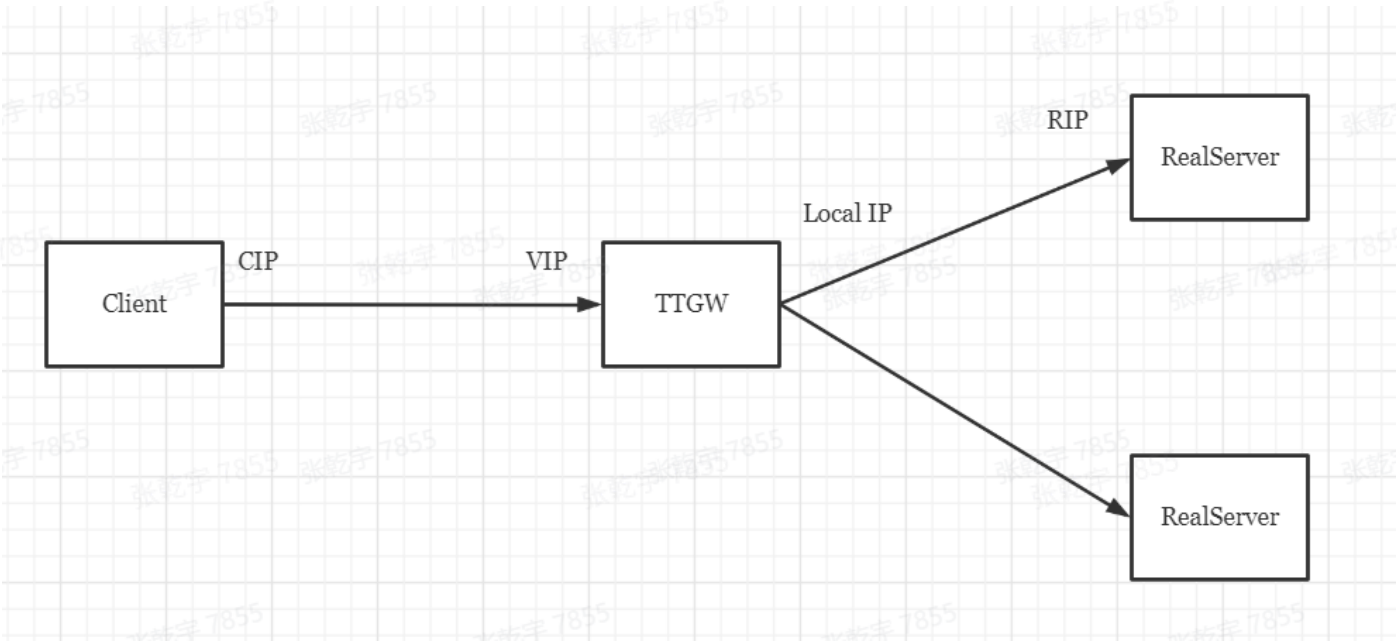
TOA/UOA设计与实现

名词解释

TOA/UOA = TCP option address / UDP option address

原来的名字， 实际上现在用的是IP option

需求背景



TOA/UOA的需求主要来自于TTGW， TTGW是一个基于DPDK的四层LB。

在TTGW的fullnat模式下， TTGW会把数据包的 client ip, client port, virtual ip, virtual port 换成LB的local ip, local port和Real Server的rip, rport, 这样后面的Real Server看不到真正的client地址信息和LB对外发布的virtual IP/port信息， 但是会有业务在Real Server上需要这些信息， 因此需要一种方法来满足这个需求。

目标场景包括 4to4, 6to6, 以及6to4（外网IPv6, 内网IPv4）等， 支持的协议包括TCP和UDP。

方案选择

以下简述可能的各种方案及其对应的优势与问题。

1. 放在TCP option里面

   优点： 对业务影响最小

   问题： 不支持UDP， TCP option容量不够， 不足以同时容纳cip, cport, vip, vport

2. 放在IP option里面

   优点： 同时支持TCP和UDP

   缺点： 某些交换机转发带IP option的包速率较低， 不能满足需求

3. 在IP 和 TCP之间设置一个新的协议

   优点： 没有容量限制

   缺点： 影响TCP的RSS, 影响TCP checksum offload

4. 在TCP 和 HTTP之间设置一个新的协议

   优点： 与操作系统耦合较低

   缺点： 影响TLS，需要业务修改代码

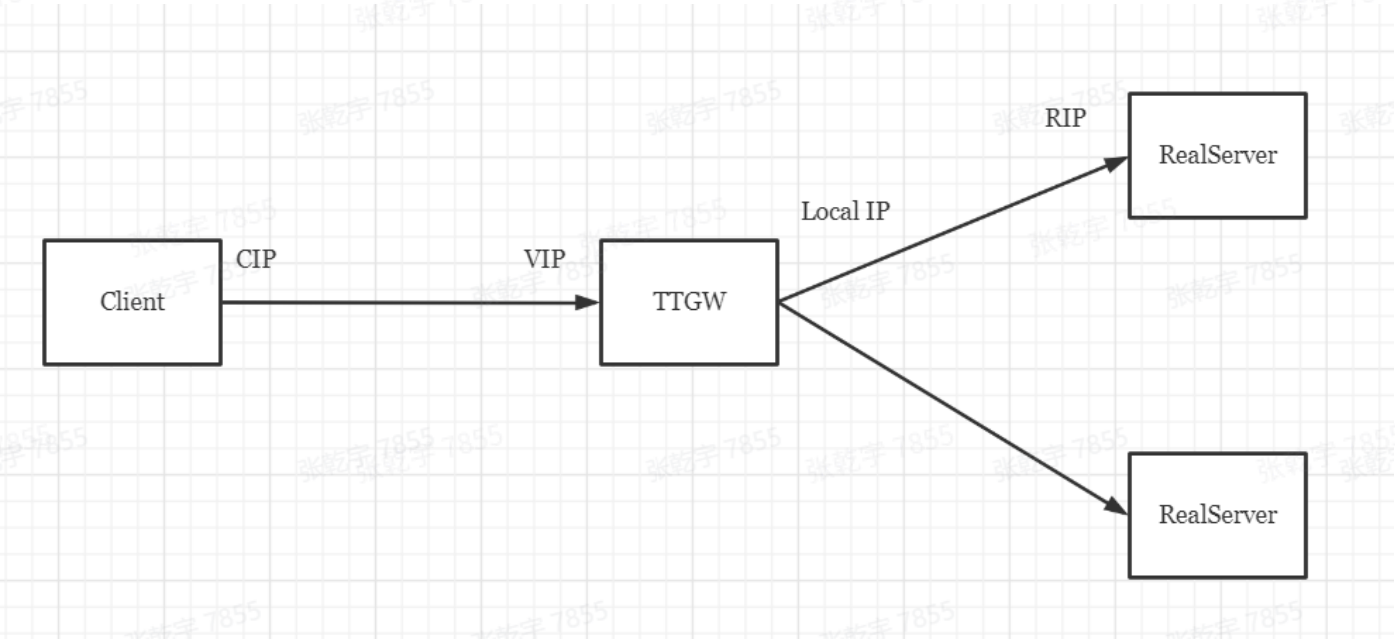
5. IP option + vxlan隧道

   优点： 没有交换机速率的问题

   缺点： 需要加一个解vxlan的模块， 需要ovs支持

之前我们得知交换机升级之后可以做到 IP option包的线速转发， 所以我们选取的是IP option的方案。后来得知交换机升级非常困难， 所以选取了IP option + vxlan的方案。IP option相关功能由新版TOA/UOA实现， vxlan相关功能由XRS/XRT实现。

部署模式



TOA/UOA和XRS/XRT会对接TTGW2.0。

TOA/UOA内核模块和XRS/XRT模块将部署在后端所有的Real Server上。其中TOA/UOA内核模块工作在网络协议栈的IP层， XRS/XRT模块通过epbf和XDP技术工作在网卡的驱动层。

协议设计

设计目标是在ip option种同时携带cip, cport, vip, vport, 且同时支持4to4, 6to6, 6to4等场景。

IPv4 option 设计

```
1
2  +--------+--------+--------+--------+
3  |                                   |
4  +--------+--------+--------+--------+
5  |  custom ipv4 header(5 * 4 bytes)  |
6  +--------+--------+--------+--------+
7  |  no specific change except hdrlen |
8  +--------+--------+--------+--------+
9  |                                   |
10 +--------+--------+--------+--------+
11 |                                   |
12 +--------+--------+--------+--------+
13 |option  |length  |operation|padding|
14 +--------+--------+--------+--------+
15 |      CPORT       |      VPORT      |
16 +--------+--------+--------+--------+
17 |               CIP                 |
18 +--------+--------+--------+--------+
```

```
|                VIP                |
+--------+--------+--------+--------+
```

Type: 31

length: IPv4 payload: (1 + 1 + 1 + 1 + 2 + 2 + 4 + 4) = 16

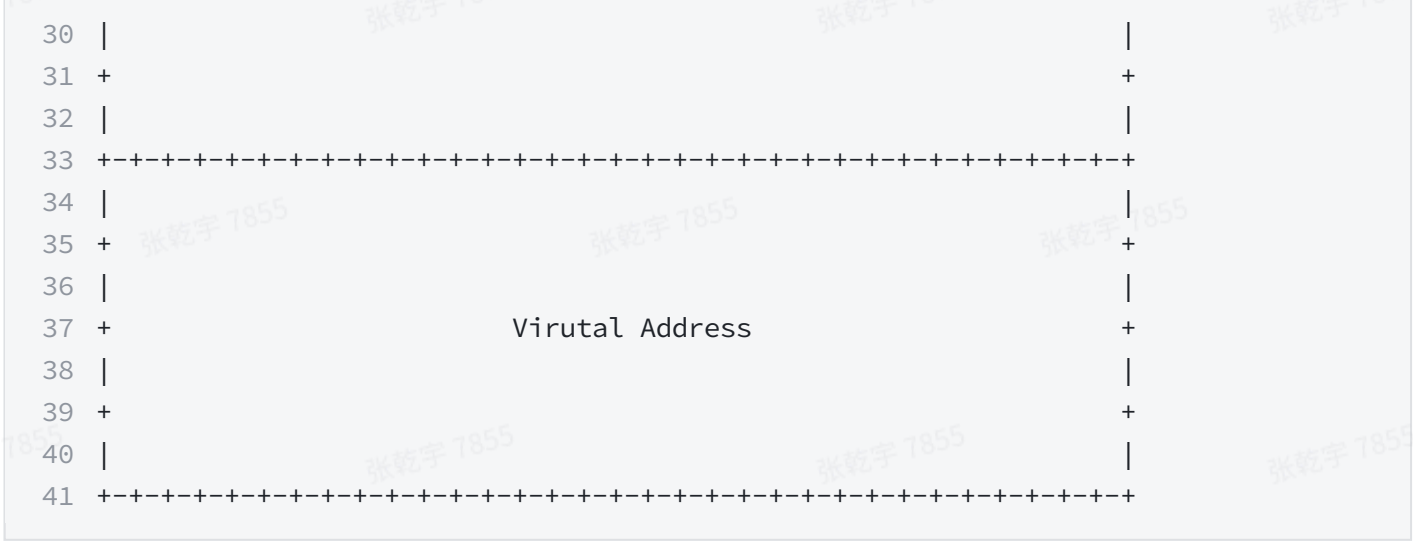
IPv6 payload: (1 + 1 + 1 + 1 + 2 + 2 + 16 + 16) = 40

Operation: the lowest bit indicates the option payload is IPv4(0) or IPv6(1).

Padding: 0

IPv6 option 设计

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  NextHeader0  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  NextHeader1  | Header Length |  OptionType   | OptionLength  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         CPort                 |             VPort             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Client Address                        +
```



在IPv6协议的规范中， ipv6头部中的next header表明后面的option header的类型, 而option header中的next header表明跟在后面的TCP协议的类型。

NextHeader0 = 60 (Destination Option)

NextHeader1 = 6 (TCP)

Header length: IPv6 payload: ((1 + 1 + 1 + 1 + 2 + 2 + 16 + 16) / 8 - 1) = 4

IPv4 payload: ((1 + 1 + 1 + 1 + 2 + 2 + 4 + 4) / 8 - 1) = 1

OptionType = 31

OptionLength: ipv6 payload: （2 + 2 + 16 +16） = 36

ipv4 payload: (2 + 2 + 4 + 4) = 12

ipv6 rfc:https://tools.ietf.org/html/rfc2460

TOA/UOA ip option整体设计

```
union two_addr{
    struct{
        unsigned char saddr[4];
        unsigned char daddr[4];
    }ipv4;
    struct{
        unsigned char saddr[16];
        unsigned char daddr[16];
```

```
    }ipv6;
};

struct ip_option{
    union{
        struct{
            __u8 type;
            __u8 length;
            __u8 operation;
            __u8 padding;
        }ipv4;
        struct{
            __u8 nexthdr;
            __u8 hdrlen;
            __u8 option;
            __u8 optlen;
        }ipv6;
    }header;

    __be16 sport, dport;

    union two_addr addrs;
};

#define IPV4_OPTION_TYPE 31
#define IPV6_HEADER_DST 60
#define IPV6_HEADER_OPTION 31

```

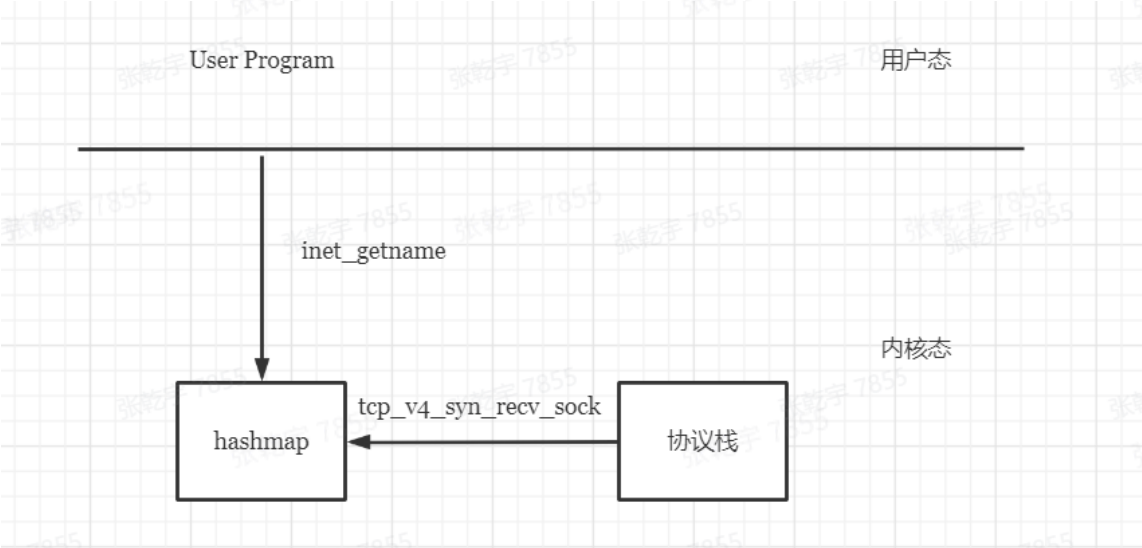
其中头部4字节根据使用的是ipv4 option或ipv6 option有所变化。后面的空间存储携带需要的cport, vport, cip, vip。 整个结构的长度根据内部负载的是ipv4 或ipv6地址而有所变化。

port 和 ip地址都使用网络字节序（大端序）。

因为 TOA/UOA 的ip option协议与TCP/UDP 无关， 且option头部支持两种外层协议（ipv4/ipv6），option尾部支持两种类型（ipv4/ipv6）的地址内容， 所以该协议同时支持2 * 2 * 2 = 8种场景。

模块设计

TOA模块

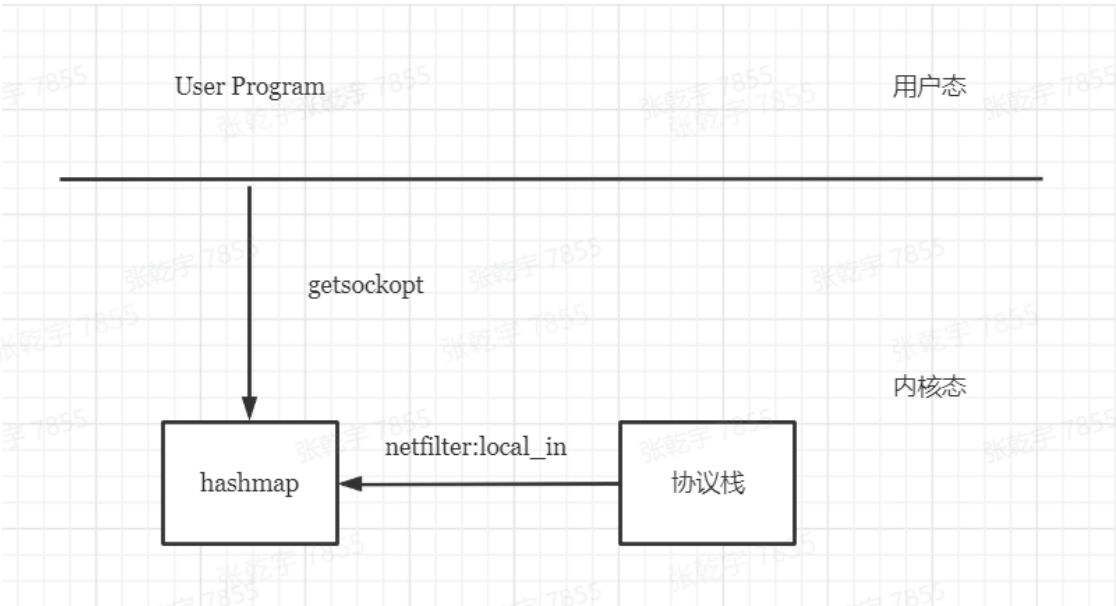


TOA模块首先维护一个hashmap, 因为tcp sock有自己的连接信息，所以该hashmap以 struct sock为key, 以 cip,cport, vip, vport为值。

TOA模块劫持了内核网络协议栈的tcp_v4_syn_recv_sock函数（ipv6同理），该函数在TCP三次握手中第三个包被server端收到时调用， 该函数调用后server端已经建立起一个代表该连接的tcp sock。TOA用一个新的函数替换掉原有的tcp_v4_syn_recv_sock函数， 在新函数中首先调用原有的tcp_v4_syn_recv_sock, 然后从数据包的ip option中解析出对应cip,cport, vip, vport并存入hashmap中。

TOA同时劫持的内核协议栈的inet_getname函数，当用户传入的sock与toa无关时， 调用原有的inet_getname, 有关时返回hashmap中对应的数据。

UOA模块



UOA模块的实现与TOA是相似的， 也是一个hashmap和两个钩子。

因为内核不会维护udp的连接信息， 所以UOA的hashmap是以 LB 和 RealServer的（LocalIP，LocalPort, RS IP, RS port）为key， 以（cip, cport, vip, vport）为值。

UOA使用内核协议栈中netfilter的LOCAL_IN 作为处理数据包的钩子，在钩子函数中判断数据包是否与UOA有关， 有关则将ip option中的信息解析出来存入hashmap中。

因为udp没有类似inet_getname的函数查询连接信息， 所以UOA对用户暴露的接口是通过getsockopt实现的， 具体做法是向getsockopt注册一个特殊的值并绑定一个实现函数，在该函数中查询hashmap并返回相应的结果。

set_ip_option工具

TOA/UOA模块在后期还遇到了一个新的需求，有业务希望在7层LB中使用这个模块向后端的Real Server传递cip,cport, vip, vport。如果直接的实现的话， 需要对7层LB所在的内核网络协议栈做非常大的改动， 但是Linux协议栈对外暴露了设置ip option的功能，所以可以通过一个set_ip_option的工具非常简单的将TOA/UOA的ip option信息设置到7层LB发出的包中。

用户接口

```
int set_ip_opt(...);// set the toa ip option
int del_ip_opt(...);// delete the toa ip option
```

TCP client as example

```

int fd = socket(...);

set_ip_opt(fd, ...);
// after this setup, the packet from this fd will all carry the toa ip option.

connect(fd, ...);

// before delete the ip option, client must send at least one packet for the third stage of the three-way handshake of TCP.
del_ip_opt(fd, ...);
// after this setup, the ip option on this fd will be deleted, the fd recover to a normal fd.
```

具体实现

```
int set_ip_opt(int fd, int fd_family, struct sockaddr* saddr, struct sockaddr* daddr);
```

用户需要在调用set_ip_opt时传递socket fd, socket 的family（AF_INET/AF_INET6）, 需要携带的source address和destination address。

set_ip_opt将传入的参数按照TOA/UOA协议存放到opt中，然后再调用

```
setsockopt(fd, IPPROTO_IP, IP_OPTIONS, opt, len);
```

就可以将ip option绑定到对应的socket fd上， 此后该fd发送的数据包都将携带传入的ip option。

示例

结合set_ip_opt工具和TOA/UOA模块， 即使在没有四层LB的情况下也可以搭建一个TOA/UOA的演示例子。

下面以TOA为例简单的演示一下用法。

首先下载toa的源码，并insmod

```
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1# ls
example  kmod  Readme.txt  scripts
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1# cd kmod/
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/kmod# make insmod
sudo rmmod toa
rmmod: ERROR: Module toa is not currently loaded
Makefile:40: recipe for target 'rmmod' failed
make: [rmmod] Error 1 (ignored)
sudo insmod toa.ko toa_map_table_bits=14
```

接着启动server

```
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/kmod# cd ..
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1# cd example
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/example# ls
Makefile  server_v4.cpp  server_v6.cpp  util.cpp  util.h
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/example# make server_v4
g++ -o server_v4 server_v4.cpp util.h util.cpp -pthread
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/example# ./server_v4 8192
```

然后下载ip_option工具的源码，并启动client

```
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/ip_option# ls
ip_opt  Readme.txt  script  tcp  udp
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/ip_option# cd tcp/
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/ip_option/tcp# ls
client_v4.cpp  client_v6.cpp  Makefile  util.cpp  util.h
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/ip_option/tcp# make client_v4
g++ -o client_v4 -I ../ip_opt client_v4.cpp util.h util.cpp ../ip_opt/ip_opt.h ../ip_opt/ip_opt.c -pthread
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/ip_option/tcp# ./client_v4 127.0.0.1 8192 1
set_ip_opt success
establish connection 3 to 127.0.0.1:8192
del_ip_opt success
worker on 3
i: 0
i: 1
i: 2
```

在server端可以看到显示的连接地址

```
root@n8-019-156:/data00/zhangqianyu/WorkSpace/TOA_UOA/develop/toa1/example# ./server_v4 8192
0's connection 4 from ipv4: 10.0.0.1:1
getpeername: ipv4: 10.0.0.1:1
getsockname: ipv4: 10.0.0.2:2
worker on 4
recv_len: 64
recv_len: 64
```

可以看到实际连接应该是127.0.0.1的地址，但是server端读到的地址是10.0.0.1， 说明地址已经被toa替换掉了。