记录 Tun 透明代理的多种实现方式，以及如何避免 routing loop #57

New issue

Open taikulawo opened this issue Aug 1, 2021 · 7 comments

taikulawo commented Aug 1, 2021 • edited ⌄ Collaborator ⋯

从tun读出来后再写入tun，下次读还会将自己刚写入的packet读出来，如果设置默认路由是tun网卡，会导致死循环。下文会介绍解决routing loop的多种方法

https://www.kernel.org/doc/Documentation/networking/tuntap.txt

> How does Virtual network device actually work ?
> Virtual network device can be viewed as a simple Point-to-Point or
> Ethernet device, which instead of receiving packets from a physical
> media, receives them from user space program and instead of sending
> packets via physical media sends them to the user space program.
>
> Let's say that you configured IPv6 on the tap0, then whenever
> the kernel sends an IPv6 packet to tap0, it is passed to the application
> (VTun for example). The application encrypts, compresses and sends it to
> the other side over TCP or UDP. The application on the other side decompresses
> and decrypts the data received and writes the packet to the TAP device,
> the kernel handles the packet like it came from real physical device.

一共两个方法，read 和 write

read from tun读取数据包

write将数据包写入tun，tun直接将userspace的packet注入内核，就好像内核刚从物理网卡读取出来一样

https://github.com/gfreezy/seeker/blob/b5a1b83a24c48bb96fb26cc3d5402dd2cd7159f1/README.adoc#%E6%8C%87%E5%AE%9A-ip-%E6%88%96%E6%9F%90%E7%BD%91%E6%AE%B5%E8%85%B0%E4%B8%A3%E7%90%86

https://github.com/gfreezy/seeker/blob/b5a1b83a24c48bb96fb26cc3d5402dd2cd7159f1/README.adoc#%E5%AE%9E%E7%8E%B0%E5%8E%9F%E7%90%86

搞这些东西是好在虚拟机，搞坏了还能还原。

NAT，转发到socks server

1. tun读取ip packet，本地启动TcpServer/UduServer
2. 解析出ip src，dest 信息
3. 通过fake by ip，利用NAT，获取一个内部网段的唯一ip
4. 将此数据包的ip dest改为自身ip，并将dest port改为TcpServer 监听的端口
5. TcpServer/UDPServer收到请求，根据预设的规则进行判断，最后决定将数据发给 socks server还是直连

这里的关键是ip packet的 dest port 指向了 TcpServer/UdpServer 监听的端口，通过这种方式，让 IP packet 重新走了一遍OS自身的 tcp/ip stack（好处是借用OS自己的协议栈 shadowsocks/shadowsocks-rust#199 (comment)），同时获取到 src ip 和 dest ip

实现往往是通过 session 实现类似NAT的功能，packet走完OS自己的协议栈后，能够还原出original destination。

看下面 Go 和 Rust 的代码，代码是不是非常像？

https://github.com/xjdrew/kone/blob/3a6d64a7ad7435134b6039129bd4ec8c5e9fbda5/k1/tcp_relay.go#L144
https://github.com/willdeeper/seeker/blob/d575fe35b55439f207e5b454bbaa3e0da8f87fcd/tun_nat/src/lib.rs#L28

类似还有最近（2021/08/10）shadowsocks-rust 搞的tun模式

随便在tun网段bind某个端口 tcp_daddr
https://github.com/shadowsocks/shadowsocks-rust/blob/4287c2aab8f1826446b68888f58462b4342a0a53/crates/shadowsocks-service/src/local/tun/tcp.rs#L67

拦截流量后查找connection，找到就将 dest 改为 tcp_daddr，这样我们不再需要自己的userspace tcp/ip stack，而是复用OS自身的网络栈。
https://github.com/shadowsocks/shadowsocks-rust/blob/4287c2aab8f1826446b68888f58462b4342a0a53/crates/shadowsocks-service/src/local/tun/tcp.rs#L155

直接从ip packet构建TcpListener

leaf使用自己修改的lwip，从ip packet直接封装出 TcpListener，这里的难点是，TcpListener往往dest port是自身，而无法获取真正的 dest port，另外TcpServer只能收到bind的端口的请求。

leaf修改lwip代码后，使得往意端口的流量都会流经TcpListener

https://github.com/willdeeper/leaf/blob/84f37a9cc348be2a2661a7f236a7628290835ddb/leaf/src/proxy/tun/netstack/tcp_listener_impl.rs#L57

现在clash和其他很多项目都用到leaf作者封装的lwip，开发出来的go-tun2socks

leaf也有nat，但只给UDP用到，主要是fake by ip
https://github.com/willdeeper/leaf/blob/84f37a9cc348be2a2661a7f236a7628290835ddb/leaf/src/proxy/tun/netstack/stack_impl.rs#L268

为什么leaf只有UDP用到nat_manager.rs？

应该只是觉着没必要罢了，修改的 lwip 已经能对 TcpListener 提供很好的支持，所以只对UDP做了 fake by ip，用到nat。这有区别上面说的 `NAT，转发到 socks server，leaf 不依赖 OS 自己的 tcp/ip 重组报文，也就不需要维护 session 来进行 NAT

如何避免重新读出刚写入tun设备的数据包

> 请问，如果自己设置默认路由全部流量都发到tun设备，自己read出来后交给userspace网络栈处理，再将处理后的数据包写入tun，之后的read会不会重新read出来自己刚写入的包呢？
>
> 我理解会，内核根据默认路由会重新送给tun，但那样死循环了呀
>
> 我看这些利用tun的transparent proxy从未遇到过这问题，就好奇是什么原理

seeker有这句话

> == 指定 IP 或某网段走代理
> 修改路由表，将希望走代理的 IP 或者网段路由到虚拟网卡。如果使用了本机 socks5 代理，则必须确保 socks5 不会直连加入路由表的网段，否则会死循环。

设置默认路由后绝大多数流量都会到tun，为了防止自己刚写到tun包的走默认路由又回来，需要再设置条更短的路由表，发包的ip dest 用这个ip网段，这样就不会再回来。

> Active Routes:
>
> Network Destination Netmask Gateway Interface Metric

```
      0.0.0.0          0.0.0.0      192.168.3.1     192.168.3.21     25
      0.0.0.0          0.0.0.0         On-link       198.18.0.1      0
    127.0.0.0        255.0.0.0         On-link        127.0.0.1    331
   198.18.0.0      255.255.0.0         On-link       198.18.0.1
```

clash为默认网关，clash使用wireguard，在windows上使用 SO_UNICAST_IP 绑定 outgoing 网卡流量，不会再走routing decision

透明代理的tcp listener都是寄生在lwip之上，从tun设备收到数据写入userspace stack，自己的socks代理直接从lwip处理后的数据里就能解析出socket，leaf就这么做的。

net stack写入tun

https://github.com/willdeeper/leaf/blob/0dff09c9bb652c53e197066af863a6f6a083ecff/leaf/src/proxy/tun/inbound.rs#L113

tun写入 net stack
https://github.com/mellow-io/go-tun2socks/blob/6289c4be8d7c2915a73ba0d303d71a3a135e1574/cmd/tun2socks/main.go#L199

https://github.com/mellow-io/mellow/blob/f71f6e54768ded3cfcc46bebb706d46cb8baac08/src/helper/linux/config_route#L1

```
# ip
CMD=$1
# tun gateway
TUN_GW=$2
# 原来的默认网关
ORIG_GW=$3
# 原来的默认网关发给的接口
# https://github.com/mellow-io/mellow/blob/f71f6e54768ded3cfcc46bebb706d46cb8baac08/src/main.js#L866
ORIG_GW_SCOPE=$4
# send through 策略路由
# https://man7.org/linux/man-pages/man8/ip-rule.8.html
# https://github.com/mellow-io/mellow/blob/f71f6e54768ded3cfcc46bebb706d46cb8baac08/src/main.js#L719
ORIG_ST=$5

"$CMD" route del default table main
# 注意，这加到了 main table 是main默认使用的路由表
"$CMD" route add default via $TUN_GW table main
# 注意，这里加到default table，default table是默认路由表，main路由表未匹配到会使用default
"$CMD" route add default via $ORIG_GW dev $ORIG_GW_SCOPE table default
# 策略路由，从原来的default interface 接口来的ip数据包转发到default表，这样就能送出去流量，不会成环了
"$CMD" rule add from $ORIG_ST table default
~~看来 wireguard 写入 tun 后并不会出现routing loop，SO_NOTOIF opt 确保不会重新读出刚才写入的数据包~~
~~https://www.wireguard.com/netns/~~
```

wireguard 之前想添加 `SO_NOTOIF` 来避免routing loop，https://lists.openwall.net/netdev/2016/02/02/222
但看来维护者认为 ip rule 就够用，不应该在内核做太多magic的功能（直接不走路由）。最后不了了之

wireguard实现
https://github.com/WireGuard/wireguard-go

mellow-io/mellow#310

http://www.policyrouting.org/iproute2.doc.html#ss9.6

防止routing loop的方式

为需要直连的ip设置单独的路由(删除掉默认路由)

> 同学，我又来了[无辜笑]。我看seal的全局模式可以转发整个系统的流量，最近自己在搞一个类似的透明代理工具，有个问题想请教下
>
> 如果自己设置默认路由全部流量都发到tun设备，自己read出来后交给userspace网络栈处理，再将处理后的数据包写入tun，之后的read会不会重新read出来自己刚写入的包呢
>
> 我理解会，内核根据默认路由会重新送给tun，但那样死循环了呀
>
> 但我看seal的全局模式transparent proxy从未遇到过这问题，这是怎么做到的呢
>
> 回复：read出来不会直接写入tun，是重新封包发给VPN server

vpn server是单独加了一条路由，而且发送包不是直接写tun包，是直接通过tcp/udp发出，socket可以绑定指定网卡发送

```
ip route del default
ip route add default dev wg0
ip route add 163.172.161.0/32 via 192.168.1.1 dev eth0
```

The Classic Solutions

为直连ip设置单独路由（不删除默认路由，只覆盖）

默认路由的作用是没有匹配到时走default，通过设置 `0.0.0.0/1`，让这条路由总是先于 default 命中。
再对要直连的 ip(这里是 163.172.161.0) 设置单独的路由，不需要删除原来的默认路由

```
ip route add 0.0.0.0/1 dev wg0
ip route add 128.0.0.0/1 dev wg0
ip route add 163.172.161.0/32 via 192.168.1.1 dev eth0
```

这种也叫做 `0/1 128/1 trick`。

但这trick有局限搜0/1

推荐阅读

Overriding The Default Route

iptables

iptables REDIRECT
https://github.com/iamwwc/ooproxy

iptables TPROXY
iptables with fwmark
iptables 配合 fwmark。
https://flylib.com/books/en/2.783.1.50/1/

策略路由 (ip rule with fwmark)

使用 ip rule 排除掉某个网卡流量
Rule-based Routing https://www.wireguard.com/netns/#routing-all-your-traffic

ip rule https://man7.org/linux/man-pages/man8/ip-rule.8.html

https://www.reddit.com/r/WireGuard/comments/m8jwnt/i_dont_understand_how_wgquick_adds_routes/grkomnp?utm_source=share&utm_medium=web2x&context=3

或者 rule 还有更强大的终止routing decision
这方法没用过，但看着能行

tailscale/tailscale#144

namespace solution

https://superuser.com/questions/1664065/tun-device-how-to-avoid-routing-dead-loop-when-write-a-transparent-proxy

The New Namespace Solution

bind before connect

bind之后connect，routing 不会起作用，这样就能解决设置默认网关后导致的 routing loop

通过调试 leaf 我已经能够十分确认上面这句话的正确性

> If I bind an interface before to connect, Does that mean the connect for outgoing traffic will use that interface I bind without follow the routing decision?
>
> @nuclear yes, if you bind() to an interface before connect()'ing, the connection will go out through that interface. That is the whole point.

listen之前需要bind，决定listen到哪个网卡，
如果作为client去connect，在调用connect时bind会隐式发生。你也可以主动bind before connect，绕过路由选择，强迫出流量使用某个network interface

https://stackoverflow.com/a/4297381/7529562

windows 平台

IP_UNICAST_IF
wireguard也依赖tun device，但这tun和Unix上的tun不太像。

windows并没有tun的概念，为弥补这空缺，wintun 既是个 windows 内核驱动，也是个userspace tunnel，前者从内核拉取、向内核注入packet，后者将前者的数据包传给 userspace 处理
https://git.zx2c4.com/wireguard-windows/about/docs/attacksurface.md#wintun。

windows 并没有策略路由，所以通过 bind interface IP_UNICAST_IF 来避免routing loop（从侧面印证了 bind before connect 可以解决routing loop）

以下三封邮件解释了为什么使用 bind。

https://lists.zx2c4.com/pipermail/wireguard/2019-September/004493.html
https://lists.zx2c4.com/pipermail/wireguard/2019-September/004541.html
https://lists.zx2c4.com/pipermail/wireguard/2019-September/004542.html

其中 https://lists.zx2c4.com/pipermail/wireguard/2019-September/004541.html 谈到的 IP_UNICAST_IF IPV6_UNICAST_IF 可理解为指定 outgoing 流量的网络接口。

而里面说的 WFP 是 Windows Filtering Platform，看起来是 windows 平台内核级别的流量过滤API（类似 iptables）

windows 没有Linux 的类似 ip rule 这种策略路由。wireguard-windows 用来 IP_UNICAST_IF 来开发

下面是 wireguard-go 在 windows 上用 IP_UNICAST_IF 的实现

wireguard-go IP_UNICAST_IF的实现

https://github.com/WireGuard/wireguard-go/blob/5846b622837e04dbc35b153d9ceda7fd66397520/conn/bind_windows.go#L567

31是windows header里定义的

https://www.pinvoke.net/search.aspx?search=IP_UNICAST_IF&namespace=[All]

此外，wireguard 会帮你自动设置路由，bind 来避免routing loop

https://www.reddit.com/r/WireGuard/comments/m8jwnt/i_dont_understand_how_wgquick_adds_routes/

wireguard 对自己使用 IP_UNICAST_IF 的解释

https://git.zx2c4.com/wireguard-windows/about/docs/netquirk.md

VPN service 直接支持设置绕过VPN的路由
https://docs.microsoft.com/en-us/uwp/api/windows.networking.vpn.vpnrouteassignment.ipv4inclusionroutes?view=winrt-22621

Linux 平台

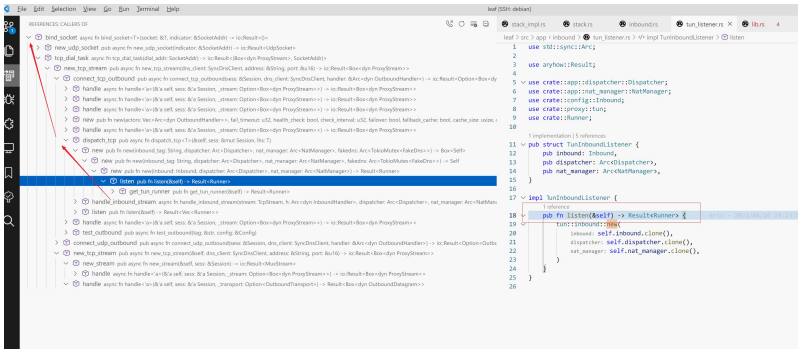
通过 setsockopt syscall 时传递 SO_BINDTODEVICE

https://stackoverflow.com/questions/4584908/how-do-i-send-udp-packet-from-a-specific-interface-on-linux

https://lore.kernel.org/netdev/1328685717.4736.4.camel@edumazet-laptop/T/

阅读leaf代码时确认使用过上述选项

https://github.com/willdeeper/leaf/blob/0dff09c9bb652c53e197066af863a6f6a083ecff/leaf/src/proxy/mod.rs#L235



shadowsocks-rust 新添加的 tun mode 也使用了

```
https://github.com/shadowsocks/shadowsocks-rust/blob/4287c2aab8f182644b68888f58462b4342a0a53/bin/aslocal.rs#L116

https://github.com/shadowsocks/shadowsocks-rust/pull/586#issuecomment-899196940
```

经测试(leaf只bind，不添加 ip rule)，bind SO_BINDTODEVICE 之后再将数据写入socket，数据会直接到达网卡的发送队列，不会再次走routing decision。

路由选择的核心在于找到一个网卡，并 bind SO_BINDTODEVICE 直接绕过了这一步。

对于从外界接收数据，

```
user mode tcp/ip stack <=> Application
                ^
                | default routing
outside => routing decision => NIC adaptor => tcp/ip stack behind NIC
                                                    \
                                                      Application
                                                    /
        (ARP)
outside <= NIC adaptor <= routing decision <= tcp/ip stack behind NIC
            ^
            |bind SO_BINDTODEVICE
user mode tcp/ip stack <=> Application
```

当 leaf 使用bind到默认网卡时，不需要ip rule策略路由。

即使socks代理在本地，也不会导致routing loop。

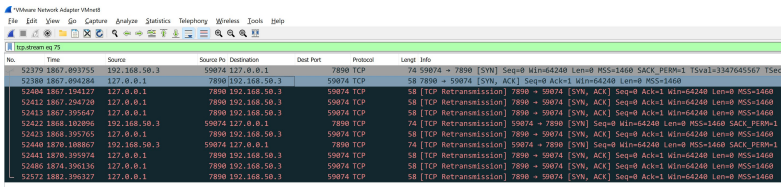
这是因为路由表中 local 表优先级最高，我们修改的是main和default表。

配置 socks outbound 为其他机器时正常工作，是因为流量到了原始网卡直接发离机器了。

但如果配置的socks outbound 为本地环路地址会又问题

例子

> 本地开clash，listen 7890，leaf配置 `socks outbound 127.0.0.1:7890`
> socket SO_BINDTODEVICE 到 eth0，用这 socket dial to 127.0.0.1:7890
> 在vmm8，也就是虚拟机的默认网卡上抓到 ip dst 为 127.0.0.1 的流量，127.0.0.1属于环路地址，不应该在以太网卡上抓到，所以连接失败。



clash这issue https://github.com/Dreamacro/clash/issues/135 谈到了bind，listen的问题

Mac

Network Extension
excludedRoutes

https://developer.apple.com/documentation/networkextension/neipv6settings/1406294-excludedroutes

wireguard 就使用到了 includedRoutes
https://github.com/WireGuard/wireguard-apple/blob/23618f994f17d8ad8f2f65d79b4a1e8a0830b334/Sources/WireGuardKit/PacketTunnelSettingsGenerator.swift#L117

https://www.v2ex.com/t/590555#r_9004049

设置路由表
Android
https://developer.android.com/reference/android/net/VpnService#protect(java.net.Socket)

```
Protect a socket from VPN connections. After protecting, data sent through this socket will go directly to the underlying network, so its traffic will not be forwarded through the
```

每个OS都有自己的特色，Linux下提供 SO_BINDTODEVICE。而Mac，Android则提供高级的API，用于解决routing loop这个问题。核心都是绕过了routing decision这项

总结

每个平台都有特殊的实现方式，

Linux

- iptables或者配合fwmark: 很多软件
- ip rule: 很多软件

Mac

- network extension: wireguard-apple
- 为出proxy单独设置路由: 经典VPN实现，VPN server往往有公网IP

Android

- VpnService#protect: shadowsocks-android

Windows

- IP_UNICAST_IF: wireguard-go
- Windows.Networking.Vpn: Maple

而又有最后相似的，古典的(classic)处理方式：
为proxy创建单独的route，避免 routing loop. https://www.wireguard.com/netns/#the-classic-solutions

参考的开源项目

C实现的tun2socks
https://github.com/russdill/tunsocks

Go实现的tun2socks，支持windows
https://github.com/Intika-Linux-Network/Tun-2-Socks

Rust实现的透明代理工具
https://github.com/willdeeper/leaf

https://stackoverflow.com/questions/14697963/tcp-ip-connection-on-a-specific-interface

Tcp connection 使用特定interface送出去流量
或者使用**策略路由**
https://www.reddit.com/r/golang/comments/4x277h/dial_a_tcp_connection_from_a_specific_interface/d6cfr4d?utm_source=share&utm_medium=web2x&context=3

还是这张图

```
user mode tcp/ip stack <=> Application
                ^
                | default routing
outside => routing decision => NIC adaptor => tcp/ip stack behind NIC
                                                    \
                                                     Application
        (ARP)                                          /
outside <= NIC adaptor <= routing decision <= tcp/ip stack behind NIC
                ^
                |bind SO_BINDTODEVICE
user mode tcp/ip stack <=> Application
```

通过bind可以将数据送给eth0(Mac地址mac0)，现在eth0怎么往下送呢？

如果ip dest(ip0)是本机另一个网卡 eth1(Mac地址mac1)，怎么才能送过去呢？

这就涉及到ARP，写入eth0后，发送ARP确定 ip dest对应的链路层地址

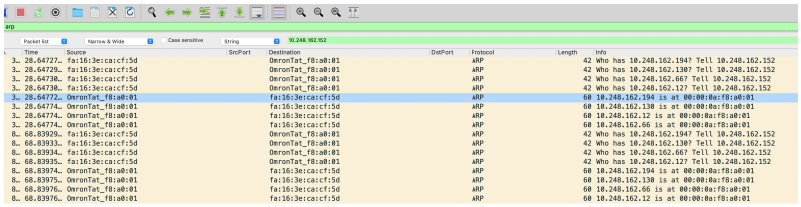
```
who has ip0 tell mac0
ip0 is at mac1
```

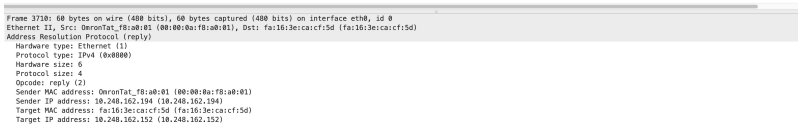
这时 eth1 告诉 eth0 MAC 地址，然后 eth0 就能送过去了。

首先确定 dest ip 是否和 src ip 为一个网段

1. 是一个网段，直接发广播ARP，链路层Target Mac全为0
2. 不是一个网段，链路层 Target Mac为网关Mac，网关收到数据包后广播再次发Arp，寻找目标地址，找到后发给他

https://zhuanlan.zhihu.com/p/370507243





如果是本地另一个网段的地址呢？

Windows 的route table 有On-Link，Linux应该也差不多。多个虚拟网卡都在内核注册过，所以内核知道 dest ip 的 Target Mac应该是哪个网卡的Mac，不需要额外发ARP，直接内核中转过去了。

这也解释了，为什么没有抓到查询另一个本地网卡的ARP请求，也解释了，从一个网卡发出去，并不是一定是发出本地机器，有可能在本地。所以leaf中bind即使是eth0，该去另一个网卡的流量还是会过去。

https://superuser.com/a/60104/944262

我很有必要区分出物理网卡，虚拟网卡。

物理网卡负责将数据从网线送出去，并没有ip地址的概念，因为不同的笔记本IP是可自由配置的，所以无法给物理网卡固定的ip。

但物理网卡是有Mac地址的。

附录

wireguard 架构
https://github.com/WireGuard/wireguard-rs

wireguard white paper 搜 routing loop

https://www.wireguard.com/papers/wireguard.pdf

6

taikulawo added the net label Aug 1, 2021

taikulawo changed the title ~~聊聊Tun 透明代理的多种实现方式，以及如何避免 routing loop~~ 记录 Tun 透明代理的多种实现方式，以及如何避免 routing loop May 13, 2022

lemos1235 commented Jun 26, 2022 • edited

话说 mac 系统上怎么处理的呢？

taikulawo commented Jun 27, 2022 Collaborator Author

> 话说 mac 系统上怎么处理的呢？

1. bind before connect
2. 单独为proxy server设置一条路由

lemos1235 commented Jun 28, 2022

还有一个问题，使用 bind 的方式，当 outbound 为本地环路地址，导致的路由循环问题还是没有解决方案吗？

taikulawo commented Jun 28, 2022 Collaborator Author

> 还有一个问题，使用 bind 的方式，当 outbound 为本地环路地址，导致的路由循环问题还是没有解决方案吗？

为啥还会有loop呢，你都bind了，我理解bind之后不会再走routing decision了

lemos1235 commented Jun 28, 2022

> > 还有一个问题，使用 bind 的方式，当 outbound 为本地环路地址，导致的路由循环问题还是没有解决方案吗？
>
> 为啥还会有loop呢，你都bind了，我理解bind之后不会再走routing decision了

那是我理解错了，。所以如果bind 以太网卡后，如果目标地址是本地的代理，是直接失败是吗？

taikulawo commented Aug 18, 2022 Collaborator Author

> > > 还有一个问题，使用 bind 的方式，当 outbound 为本地环路地址，导致的路由循环问题还是没有解决方案吗？
> >
> > 为啥还会有loop呢，你都bind了，我理解bind之后不会再走routing decision了
>
> 那是我理解错了，。所以如果bind 以太网卡后，如果目标地址是本地的代理，是直接失败是吗？

我说的不太对，mac上通过syscall做透明代理不太行，你看看这个 https://chaochaogege.com/2021/08/01/57/#Mac

通过 network extension 的includedRoutes 和 excludeRoutes 实现

eli-yip commented Feb 8, 2024

能否这样理解，对于linux上的clash tun(开启auto-route)而言，所有程序(除去clash自身)的出站流量会通过 `ip rule` 转发到 `tun` 网卡，然后clash对其进行处理，对于需要向外网发送的请求，clash通过绑定网卡(或者说绑定IP)的方式，避免被 `ip rule` 将流量重定向到 `tun` 网卡造成回环。同时也因为clash是绑定网卡的，因此查看 `ip rule` 并不能看到clash的向外流量直接发出(而不经过 `tun` 网卡)的规则？

Assignees
No one assigned
Labels
net
Projects
None yet
Milestone
No milestone
Development
No branches or pull requests

3 participants