哈爾濱工業大學

毕业设计（论文）中期报告

题 目：基于 dpdk 的高性能的 IPSec VPN

专　　业　信息安全

学　　生　殷悦

学　　号　**150120526**

班　　号　**1504201**

指导教师　刘扬

日　　期　**2019 年 4 月 28 日**

1．论文工作是否按预期进行、目前已完成的研究工作及结果

1.1 论文工作是否按预期进行

已经按照预期完成的进行，已完成内容如下:

1. 完成 strongswan 的搭建，strongswan 用于完成 ike 协商，支持 ikev1 和 ikev2

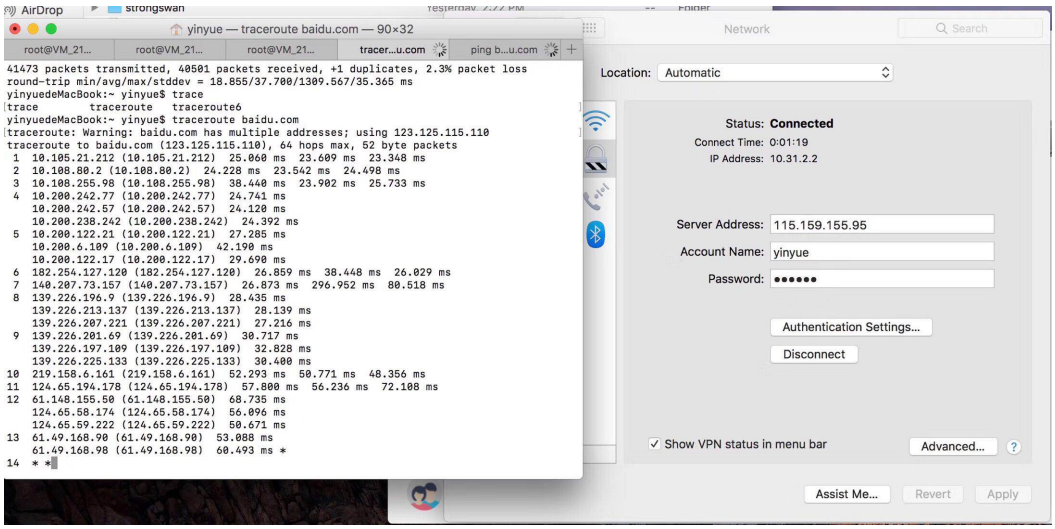

图 1-1 连接 IPSec VPN 后的界面

2. 成功使用 libnl 的 xfrm 协议监听到 ipsec vpn 在完成 ike 协商后传向内核的 sa 和 sd

图 1-2 xfrm 程序监听到 ipsec vpn 发往内核的 sa 和 sp

3. 完成 kni 网卡的设计，实现了网卡数据到达 DPDK 用户态后，由 DPDK 用户态驱动将数据转交给内核网络协议栈

```
[root@localhost bin]# ./dserver -c 0xf -n 1 -- -p1 --config '(0,0,1,2,3)'

dpdkdns version: 0.1.3
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:01:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:03:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:06:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:07:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:08:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
APP: Initialising port 0 ...
KNI: pci: 03:00:00       8086:1539

Checking link status
.............................done
Port 0 Link Up - speed 1000 Mbps - full-duplex
APP: Lcore 1 is writing to port 0
APP: Lcore 2 is working to port 0
APP: Lcore 3 is sending to port 0
APP: Lcore 0 is reading from port 0
APP: Configure network interface of 0 up
APP: Configure network interface of 0 up
port:0  rx:271 p/s  42001 bytes/s  tx:186 p/s  17784 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:2 p/s  1138 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
port:0  rx:1 p/s  74 bytes/s  tx:1 p/s  74 bytes/s  dropped:0
```

图 1-3 DPDK kni 程序收到其他设备的 ping 包并转发至内核

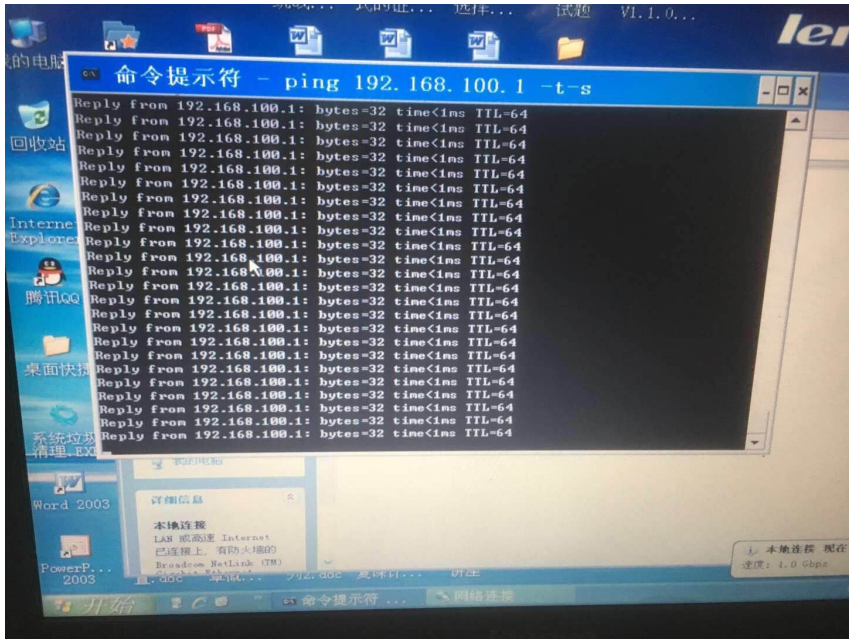

图 1-4 成功将数据发给 DPDK kni 程序并被处理

4. 完成 DPDK 程序数据分流，将 ESP 数据分流出来，将 ike 及其他数据通过 kni 网卡转交给内核

```
Creating SA context with 128 maximum entries
        spi_out(  5):aes-128-cbc sha1-hmac mode:IP4Tunnel 172.16.1.5 172.17.2.5
Creating SP context with 1024 max rules
IPv4 sp_ip4_in_0 entries [1]:
        1:0.0.0.0/0 192.168.1.0/24 0 : 65535 0 : 65535 0x0/0x0 0x1-0x0-0x5
acl context <sp_ip4_in_0>@0x7ff40058bd80
  socket_id=0
  alg=2
  max_rules=1024
  rule_size=96
  num_rules=1
  num_categories=1
  num_tries=1
Creating SP context with 1024 max rules
IPv4 sp_ip4_out_0 entries [1]:
        1:0.0.0.0/0 192.168.1.0/24 0 : 65535 0 : 65535 0x0/0x0 0x1-0x0-0x5
acl context <sp_ip4_out_0>@0x7ff400571f40
  socket_id=0
  alg=2
  max_rules=1024
  rule_size=96
  num_rules=1
  num_categories=1
  num_tries=1
IPSEC: No IPv6 SP Inbound rule specified
IPSEC: No IPv6 SP Outbound rule specified
Creating IPv4 Routing Table (RT) context with 1024 max routes
LPM: Adding route 172.17.2.5/32 (1)
LPM: Adding route 192.168.1.0/24 (3)
Allocated mbuf pool on socket 0
-----------kni-----------
1--------------------
nb_sys_ports:4
ports_mask:1
init_kni_param:0
conf.name:vEth0
KNI: pci: 03:00:00       8086:1539
init_kni_param:1
init_kni_param:2
init_kni_param:3
-----------kni-----------
Configuring device port 0:
Address: 68:ED:A4:06:39:4B
Creating queues: nb_rx_queue=1 nb_tx_queue=2...
Setup txq=0,0,0
Setup rxq=0,0,0
Setup txq=1,1,0

Configuring device port 1:
Address: 68:ED:A4:06:39:4C
Creating queues: nb_rx_queue=1 nb_tx_queue=2...
```

图 1-5 将 KNI 程序和 DPDK 程序结合到一起

```
IPSEC: Change MTU of port 0 to 1450
XFRM_MSG_NEWSA runs
 src : 192.168.100.2             dst : 192.168.100.1
 proto : 50(esp:50 ah:51)              spi : 0xcbec60ab
 repid : 1                mode : tunnel
 replay window : 32
 hmac(sha1)     0x834636e534ebe47cd8ced0547fcfe119577e0a15
 cbc(aes)       0xde9b1fb9cea3ca933a5ccff6cf5c72dd
 sel src : 00:00:00:00:00:       dst : 00:00:00:00:00:
XFRM_MSG_NEWSA runs
 src : 192.168.100.1             dst : 192.168.100.2
 proto : 50(esp:50 ah:51)              spi : 0x8fbe6bb
 repid : 1              mode : tunnel
 replay window : 0
 hmac(sha1)     0x39dc7c04b6af07050a6bb983925f8245ba7c44d2
 cbc(aes)       0x9d81635389140545d364742a801f136e
 sel src : 00:00:00:00:00:       dst : 00:00:00:00:00:
recv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv_xfrm:sa
out 3421266091 cipher_algo aes-128-cbc cipher_key de:9b:1f:b9:ce:a3:ca:93:3a:5c:cf:f6:cf:5c:72:dd auth_algo sha1-hmac auth_key 83:46:36:e5:34:eb:e4:7c:d8:ce:d0:54:7f:cf:e1:19:57:
7e:0a:15 mode ipv4-tunnel src 192.168.100.2 dst 192.168.100.1
out 3421266091 cipher_algo aes-128-cbc cipher_key de:9b:1f:b9:ce:a3:ca:93:3a:5c:cf:f6:cf:5c:72:dd auth_algo sha1-hmac auth_key 83:46:36:e5:34:eb:e4:7c:d8:ce:d0:54:7f:cf:e1:19:57:
7e:0a:15 mode ipv4-tunnel src 192.168.100.2 dst 192.168.100.1
status->status:0
status->parse_msg:
XFRM_MSG_NEWPOLICY runs
XFRM_MSG_NEWPOLICY runs
XFRM_MSG_NEWPOLICY runs
recv_xfrm:sa
out 150726331 cipher_algo aes-128-cbc cipher_key 9d:81:63:53:89:14:05:45:d3:64:74:2a:80:1f:13:6e auth_algo sha1-hmac auth_key 39:dc:7c:04:b6:af:07:05:0a:6b:b9:83:92:5f:82:45:ba:7
c:44:d2 mode ipv4-tunnel src 192.168.100.1 dst 192.168.100.2
out 150726331 cipher_algo aes-128-cbc cipher_key 9d:81:63:53:89:14:05:45:d3:64:74:2a:80:1f:13:6e auth_algo sha1-hmac auth_key 39:dc:7c:04:b6:af:07:05:0a:6b:b9:83:92:5f:82:45:ba:7
c:44:d2 mode ipv4-tunnel src 192.168.100.1 dst 192.168.100.2
status->status:0
status->parse_msg:
recv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagere
cv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv
 esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv e
sp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp
 packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp p
ackagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp pac
kagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packa
gerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp packagerecv esp package
```

图 1-6 成功分流出 ESP 数据包，IKE 数据包走 KNI 网卡

并通过测试，手机已经可以通过该 DPDK 程序完成连接

图 1-7 手机成功通过 kni 网卡连接上 ipsec vpn

5. 成功完成 xfrm 程序与 DPDK 程序的对接，可以将 xfrm 程序监听到的 sa 和 sd 传给 dpdk 程序

```
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
sadb:0x7f0c083b9fc0
esp idx:17 spi:2434762952
esp esp idx:3364364177 sa spi:3364364177 address:0x7f0c083bac80
IPSEC: No cryptodev: core 0, cipher_algo 5, auth_algo 12
```

图 1-8 xfrm 程序通过共享内存将 sa 信息传给 dpdk 主程序

6. 完成 aes_ni CPU 指令集的 DPDK 加密加速，但 CPU 不支持 AES NI 指令节

```
[root@localhost ipsec-secgw]# vim cmd_run2.sh
[root@localhost ipsec-secgw]# ./cmd_run2.sh
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:03:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1539 net_e1000_igb
EAL: PCI device 0000:06:00.0 on NUMA socket -1
EAL:   probe driv... 8086:153. ne... 1000_igb
PMD: Initialisin  crypto_aesni_mb on NUMA node 0
PMD:   Max numbe  of queue pairs =
PMD:   Max number ... sessions = .048
```

图 1-9 成功加载 aesni 驱动

7. 使用 openssl 加密库，完成数据包的加解密，并通过 gdb 打印出数据包的 IP 头信息

```
process_pkts_inbound traffic->ipsec.num:1
esp idx:91 spi:3675402435
ipsec_enqueue nb_pkts:1
ipsec_dequeue nb_pkts:0 nb_cops:-15664  ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:0
not single_sa in
process_pkts_inbound traffic->ipsec.num:1
esp idx:91 spi:3675402435
ipsec_enqueue nb_pkts:1
ipsec_dequeue nb_pkts:0 nb_cops:-15664  ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:0
not single_sa in
process_pkts_inbound traffic->ipsec.num:1
esp idx:91 spi:3675402435
ipsec_enqueue nb_pkts:1
ipsec_dequeue nb_pkts:0 nb_cops:-15664  ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:0
not single_sa in
process_pkts_inbound traffic->ipsec.num:1
esp idx:91 spi:3675402435
ipsec_enqueue nb_pkts:1
ipsec_dequeue nb_pkts:32        nb_cops:32      ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:32

Breakpoint 1, process_pkts_inbound (traffic=0x7fffffffca30,
    ipsec_ctx=0x947940 <lcore_conf+9600>) at /mnt/share/ipsec-secgw/ipsec-secgw.c:470
470                     ip_dst=ip->ip_dst;
Missing separate debuginfos, use: debuginfo-install libpcap-1.5.3-11.el7.x86_64 openssl-l
ibs-1.0.2k-16.el7.x86_64 zlib-1.2.7-18.el7.x86_64
(gdb) p ip->ip_src
$1 = {s_addr = 33693450}
(gdb) p ip->ip_dst
$2 = {s_addr = 134744072}
(gdb) p ip_src
$3 = <optimized out>
(gdb) p inet_ntoa(ip->ip_src)
$4 = 0x7ffff7fe7718 "10.31.2.2"
(gdb) p inet_ntoa(ip->ip_dst)
$5 = 0x7ffff7fe7718 "8.8.8.8"
(gdb)
```

图 1-10 成功解密 esp 数据包并通过 gdb 打印出 ip 头

8. 完成 DPDK 的 ACL 访问控制列表，实现 IPSec 的 SP 部分，并实现路由，将路由表中匹配的目的地址的数据包通过指定网卡发出

```
1 rte_cryptodev_enqueue_burst(0,0,0x96ea30,1)
ipsec_enqueue nb_pkts:1
1 rte_cryptodev_dequeue_burst(0,0,0x7ffd961cf758,32) cqp->in_flight:0ipsec_dequeue nb_pkts:n
b_cops:1        ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:1
src: 10.31.2.3  dst: 8.8.4.4
proto:17        src:10.31.2.3   dst:8.8.4.4     sport:61921     dport:13568
sa_idx:17       res:17
sa_ctx->sa[sa_idx].spi:3283584401       priv->sa->spi:3283584401
4 route_pkts
lcore_id:0      len:1   qconf:0x96c480
send to port:1
drain_buffers pkts:1

1 prepare_traffic
2 send_to_kni
send_to_kni
3 process_pkts
process_pkts_inbound
process_pkts_inbound traffic->ipsec.num:3
esp idx:17 spi:2441459651
esp idx:17 spi:2441459651
esp idx:17 spi:2441459651
1 rte_cryptodev_enqueue_burst(0,0,0x96ea30,1)
1 rte_cryptodev_enqueue_burst(0,0,0x96ea30,1)
1 rte_cryptodev_enqueue_burst(0,0,0x96ea30,1)
ipsec_enqueue nb_pkts:3
3 rte_cryptodev_dequeue_burst(0,0,0x7ffd961cf748,32) cqp->in_flight:0ipsec_dequeue nb_pkts:n
b_cops:3        ipsec_ctx->nb_qps:1
process_pkts_inbound nb_pkts_in:3
src: 10.31.2.3  dst: 8.8.4.4
src: 10.31.2.3  dst: 8.8.4.4
src: 10.31.2.3  dst: 8.8.4.4
proto:17        src:10.31.2.3   dst:8.8.4.4     sport:45270     dport:13568
sa_idx:17       res:17
sa_ctx->sa[sa_idx].spi:3283584401       priv->sa->spi:3283584401
proto:17        src:10.31.2.3   dst:8.8.4.4     sport:44785     dport:13568
sa_idx:17       res:17
sa_ctx->sa[sa_idx].spi:3283584401       priv->sa->spi:3283584401
proto:17        src:10.31.2.3   dst:8.8.4.4     sport:32454     dport:13568
sa_idx:17       res:17
sa_ctx->sa[sa_idx].spi:3283584401       priv->sa->spi:3283584401
4 route_pkts
lcore_id:0      len:1   qconf:0x96c480
send to port:1
lcore_id:0      len:2   qconf:0x96c480
send to port:1
lcore_id:0      len:3   qconf:0x96c480
send to port:1
drain_buffers pkts:3
```

图 1-11 使用 ACL 实现过滤数据包和路由表功能

9. 用网线连接 DPDK 出口网口与 Linux 物理网卡，在物理网卡抓到手机通过 IPSec VPN 发出的数据包解密解封装后的结果

```
05:31:48.350333 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xb9), length 132
05:31:49.350018 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 186, length 6
4
05:31:49.350451 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xba), length 132
05:31:50.350079 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 187, length 6
4
05:31:50.350559 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xbb), length 132
05:31:51.350068 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 188, length 6
4
05:31:51.350525 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xbc), length 132
05:31:52.350078 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 189, length 6
4
05:31:52.350556 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xbd), length 132
05:31:53.350080 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 190, length 6
4
05:31:53.350541 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xbe), length 132
05:31:54.350081 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 191, length 6
4
05:31:54.350544 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xbf), length 132
05:31:55.350024 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 192, length 6
4
05:31:55.350505 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xc0), length 132
05:31:56.350019 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 193, length 6
4
05:31:56.350519 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xc1), length 132
05:31:57.350080 IP 10.31.2.254 > 10.31.2.1: ICMP echo request, id 28112, seq 194, length 6
4
05:31:57.350541 IP 192.168.100.1 > 192.168.100.2: ESP(spi=0x0a74b620,seq=0xc2), length 132
```

图 1-12 成功抓到 DPDK 解包后的数据包

10. 完成 SP 部分工作，当 DPDK 程序收到返回的数据包后，明文数据包通过目的地址匹配 ACL 规则，获得对应客户端的 spi，通过 spi 获取加密密钥和客户端真实 IP 地址，并将数据原路返回

图 1-13 响应数据包原路返回

1.2 目前已完成的研究工作及结果

1.2.1 IPSec 网关设计与实现

本程序主要涉及 IPSec 的 IKE 和 ESP 两部分，IKE 用作协商密钥和传输规则，而 ESP 用来传输数据的。IKE 又分为 IKEv1 和 IKEv2 两个版本。由于 IKE 阶段数据包较少，实现非常复杂，因此可以使用其他 IPSec VPN 实现 IKE 过程，ESP 阶段数据量巨大，实现相对容易，需要使用 DPDK 进行加速。ESP 部分通信数据加解密使用对称加密算法，若使用软加密，则加密速度成为瓶颈，可使用硬加密来提升加密速度。

由于 ESP 部分需要加密，而 IKE 部分不需要加密，当数据到达网卡后，需要对数据进行分流，ESP 部分交由 DPDK 进行处理，而 IKE 和其他数据需要传入 Linux 内核协议栈。DPDK 和 Linux 内核态协议栈交换报文有两种模式：KNI 或 TUN/TAP。KNI 比 Linux 现有的 TUN/TAP 接口速度更快，因为和 TUN/TAP 相比，KNI 消除了系统调用和其数据拷贝。本程序采用 KNI 来实现 DPDK 和 Linux 内核态协议栈通信。当数据到达网卡后，DPDK 取到数据，获取数据包 IP 头的协议类型，若是 IPv4 或 IPv6 的 ESP 协议，则交由 DPDK 继续处理，否则交由 Linux 内核进行处理。这样就完成了分流功能。

ESP 部分需要 IKE 协商的 SA 安全联盟和 SP 安全策略。在 Linux 内核 2.6 以后，内核实现了 IPSec 传输部分(ESP 和 AH)和 XFRM 框架。因此 IPSec VPN 有两种方案，第一种方案：IPSec VPN 实现 IKE 部分并使用自己的 ESP / AH 程序来处理数据包。第二种方案：IPSec VPN 实现 IKE 部分，并使用 XFRM 框架将 A 安全联盟和 SP 安全策略传给内核，由内核处理 ESP / AH 数据包。本程序实现类似内核的 ESP 部分。使用 StrongSwan 完成 IKE 部分，配置开启 StrongSwan 的 IKEv1 和 IKEv2，配置 StrongSwan 使用内核 IPSec。使用 NetLink 的 libnl 库监听 IPSec VPN 传给内核的 SA 安全联盟和 SP 安全策略，并将其传给 DPDK 的 ESP 部分程序，而内核虽然收到了 SA 安全联盟和 SP 安全策略，但是由于第一步 ESP 数据被分流给 DPDK 进行处理，内核 IPSec VPN 收不到 ESP 数据。

虽然 DPDK 高速转发性能非常出色，但是 DPDK 没有协议栈。Linux 内核使用 ARP 协议 IP 和 MAC 映射关系。而 DPDK 协议栈处理 ESP 数据包时，由于没有 ARP 协议，难以生成以太网头源 Mac 和目的 Mac。当数据的到达网卡后，IKE 部分数据通过 KNI 被分流到内核。本程序使用内核的 ARP 协议解析 MAC 地址，并解析发往内核的 IKE 数据包，解析 IKE 数据包的以太网头和 IP 头，并保存以太网中的 MAC 地址和 IP 头存入数组，用作后续 ESP 协议通过 IP 查询 MAC 地址来生成以太网头。

1.2.2 KNI 内核协议栈转发

如图所示，KNI 在内核注册一个网卡设备，常见的网卡配置工具可以直接配置该网卡信息，通过基于 FIFO 机制来同步控制信息，网络数据使用共享内存来实现。

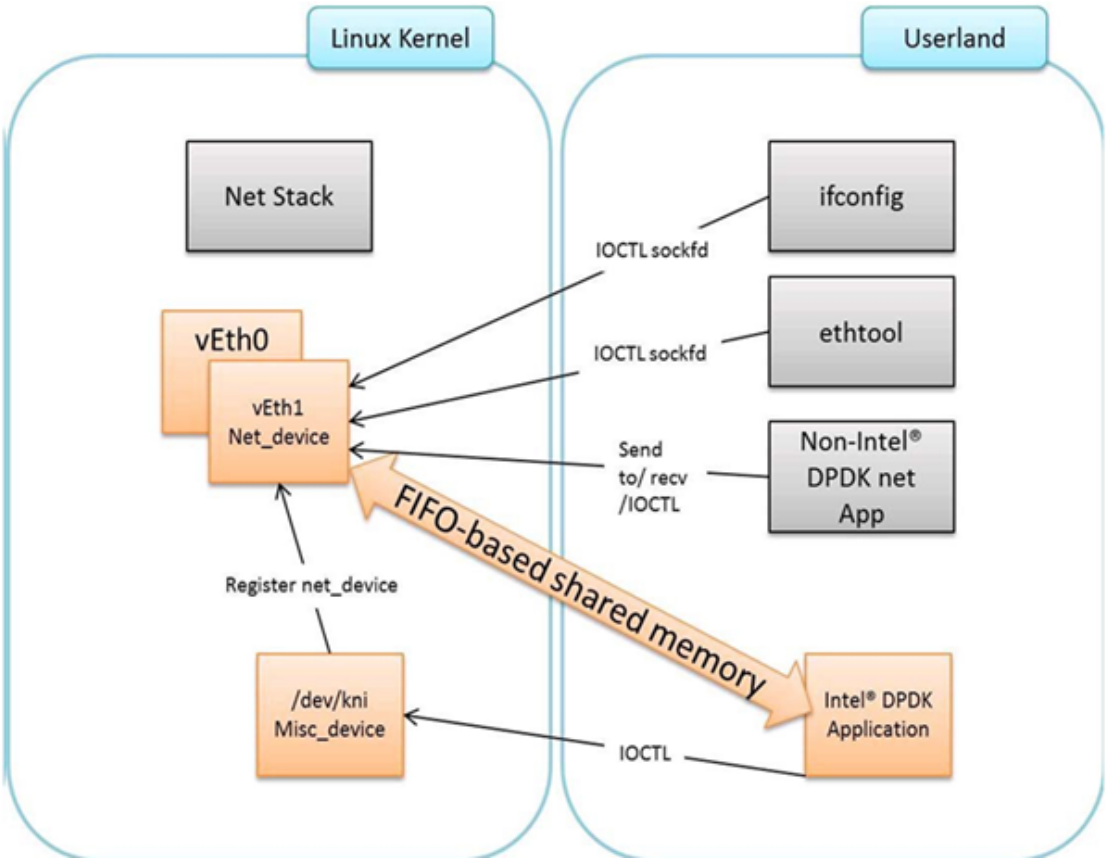

图 2-1 KNI 原理图

数据的处理流程如图

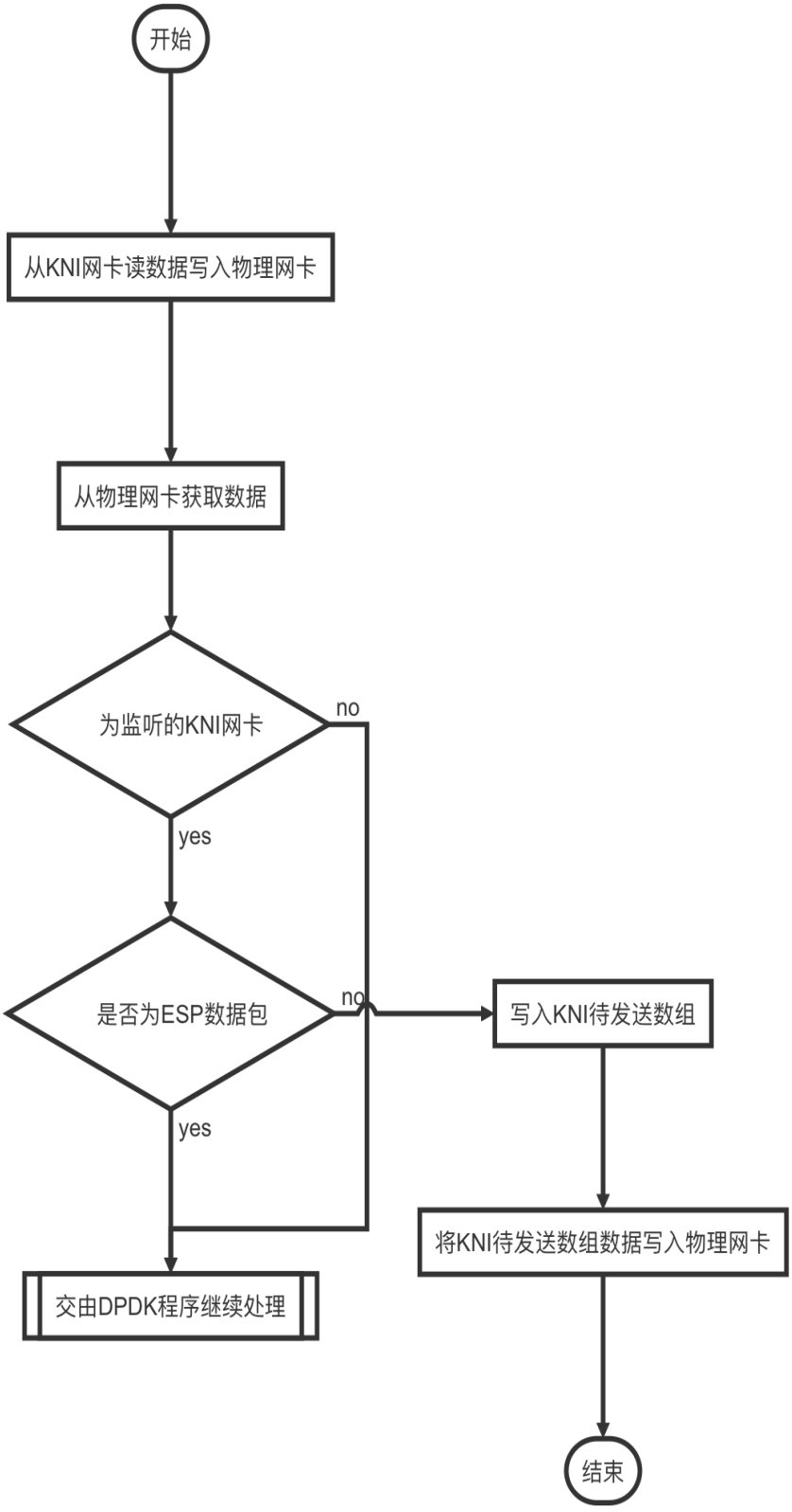

图 2-2 数据包分流流程图

1.2.3 XFRM 监听设计

DPDK 处理数据包和 XFRM 获取 IPSec 协商结果为两个并行操作，可通过多进程或多线程的方式实现并行操作。若使用多进程，进程间通信方式有：管道，信号，消息队列，信号量，共享内存，原始套接字。

在 Linux 系统中，进程是资源分配的最小单位，线程是调度的最小单位。多进程数据共享复杂，需要 IPC(进程间通信)，但数据同步复杂，而多线程数据在同一个线程中，共享简单，数据同步复杂。和多线程相比，多进程完全复制内存空间，占用空间多，切换复杂，CPU 利用率低，创建销毁速度慢，切换复杂。多进程编程和调试相对简单。和多线程比，多进程间不会相互影响，而多线程中一个线程挂掉可能导致整个进程挂掉。多进程适合多核心多机分布，扩展到多台机器比较简单，而多线程适合多核分布。结合上述有缺点，本程序只需要两个并行任务，不需要频繁创建销毁，任务间通信数据量不大，因此使用多线程。

由于进程间资源隔离，通常进程间不可互相访问。但很多情况下进程通信不可避免，需要进程通过内核或其他进程间通信来完成。常见的进程间通信应用场景有：数据传输、事件通知、共享数据、进程控制、资源共享。管道有三种：无名管道 PIPE、流管道、有名管道 FIFO。无名管道只能父子间通信，并且只能单向流动。流管道可以在父子间双向传输。有名管道可以在多个不相关进程间通信。管道是内核管理的一块缓冲区，空间不大，数据结构为环形以便重复利用。若管道中没有数据，则读取进程会一直等待，直到写进程写入数据；若管道写满，写进程会一直等待，直到读进程取出数据。当两个进程终止之后，管道也会消失。信号可以发给进程，无需知道进程状态，进程若被挂起，当进程恢复执行时才传给进程。若进程阻塞信号，信号传递讲被延迟，知道取消阻塞才继续被传递。信号是一种异步通信方式，来源有硬件和软件。套接字常用于网络间通信，也可用于进程间通信。由于管道，FIFO，消息队列等都使用内核进行通信，而系统调用是用户空间和内核空间的唯一接口，并且需用户态和内核态进行数据复制，消耗比较大，而本程序对实施性要求比较高，因此这几种方案不可取。而共享内存是通过将同一块内存区映射到不同进程地址空间中，不经过内核，因此共享内存是 IPC 中速度最快的，但共享内存需要用户来操作并且同步也需要用户来完成。因此本程序采用最复杂的 mmap 共享内存完成，并使用 CAS 无锁技术来避免加锁，提高性能。

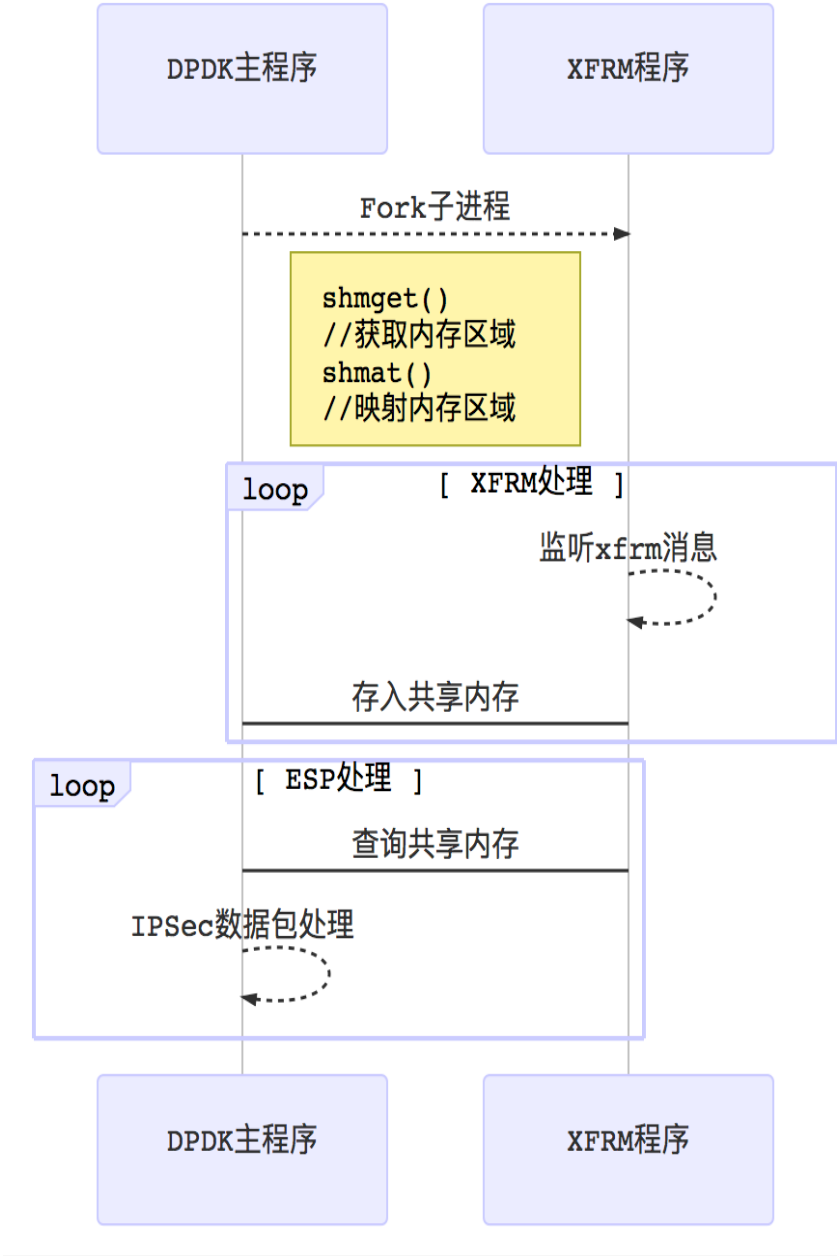

图 2-3 共享内存时序图

xfrm 使用 netlink 机制来实现 ipsec 用户态程序和内核通信。netlink 是 Linux 内核与用户空间进程通信的一种机制，类似于 UDP 协议，也是网络应用程序与内核通信最常见的接口。netlink 是一种异步通信机制，在内核和用户态之间，传递消息存不用等待存在在 socket 缓冲队列中即可，而系统调用和 ioctl 是同步通信机制。本程序采用 libnl 库来实现捕获应用程序发往内核的 ipsec 的 sa 和 sd。

用户态 VPN->内核(被捕获)

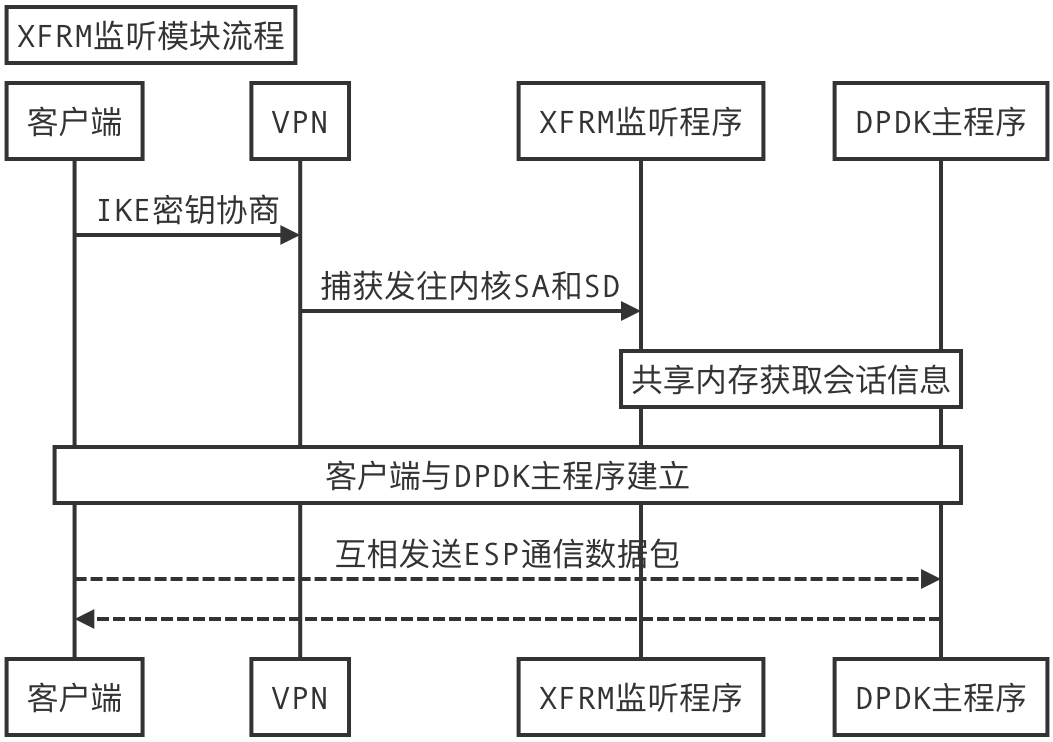

图 2-4 总程序时序图

1.2.4 IP-Mac 映射表

Mac 地址获取如图，当数据不为 ESP 数据包时，其他数据(如 ARP,ICMP,TCP,UDP)走 KNI 网卡，若数据为 UDP 协议，目标地址为 KNI 网卡 IP，端口为 500 时，此数据包为 IKE 通信数据包，可将该数据包的 IP 和端口作为一组 IP 和 MAC 映射表存入数组以备查询使用，以太网头、IP 头、UDP 头如下：

UDP 数据封装

以太网头:目的地址(6)|源地址(6)|帧类型(2)

IP 头:

版本(4)|首部长度(4)|服务类型(8)|总长度(16)|

标识(16)|标志(3)|片偏移(13)|

TTL(8)|协议(8)|首部校验和(16)|

源 IP 地址(32)|

目的 IP 地址(32)|

UDP 头:

源端口(8)|目的端口(8)|包长度(8)|校验和(8)

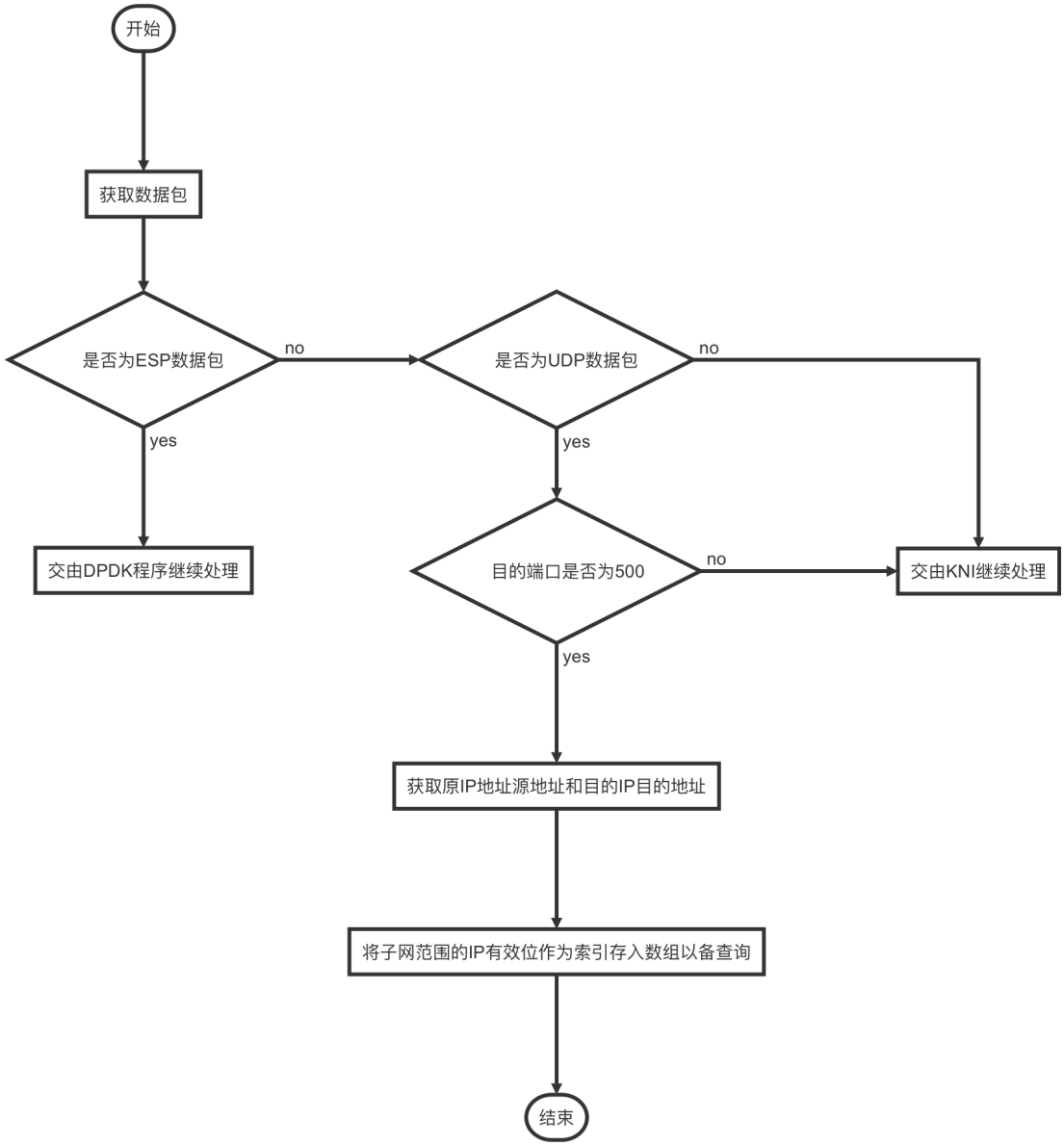

图 2-5 IP-MAC 映射表处理流程图

2．后期拟完成的研究工作及进度安排

2.1 后期拟完成的研究工作

1.完成 DPDK 数据转发到 tun 网卡

2.完成 ipsec 客户端，DPDK 程序，Linux 内核网卡的对接

3.搭建 DPDK-pktgen 发包工具环境并完成性能测试

4.如果有时间和设备，完成使用 QAT 加密卡、AES-NI CPU 指令集加密加速及其性能提升

5.使用类似内核自旋锁模型的 CAS 操作来替换多进程共享内存数据同步安全问

题

2.2 后期进度安排

5 月 6 日-5 月 10 日 完成 tun 功能

5 月 10 日-5 月 13 日 完成对接工作

5 月 13 日-5 月 20 日 完成性能测试

3．存在的问题与困难

问题 1：DPDK 程序和 pktgen 性能测试工具同时启动，程序崩溃

解决方案：机器 CPU 性能和内存不足，发包机和测试机不能使用同一台机器

问题 2：内核无法收到来自 KNI 网卡的数据

解决方案：

1. rte_kni 的驱动不带入参数，直接 insmod 即可，例子中的 insmod=lo_mode 的意思是仅仅把报文发给回环网卡而不处理，lo_mode 是 disable 状态时，数据才经过内核协议栈
2. 使用 kni 设备的 iptables 关闭，在 centos7.3 中，需要将 iptables 先启动在停止才生效，iptables 无法处理数据

问题 3：升级内核版本后 DPDK 无法启动

解决方案：DPDK 程序依赖内核驱动，DPDK 17.02 版本必须使用 Linux 3.10.0-514 版本的内核，不能随便使用 yum update 升级内核，可手动从官网下载对应版本内核的 kernel、kernel-tools、kernel-tools-libs，然后先安装旧版本内核程序，再强制卸载新版本内核

问题 4：安装 AES-NI CPU 加速加密环境后，程序崩溃

解决方案：CPU j1900 不支持 AES 加密加速 AES-NI 指令集，需更使用支持 AES-NI 指令集设备

问题 5:数据必须攒满缓冲区才会发送到物理网卡

解决方案：程序默认缓冲区满时发送数据包。可设置定时器，定期发送缓冲区中数据到物理网卡

问题 6:内核物理网卡无法接受 DPDK 物理网卡发出的数据

解决方案：将目的 MAC 地址设置为内核目的物理网卡 MAC 地址并添加 ARP 映射表，内核仍无法处理数据包，拟在后期使用将数据直接通过 tun 虚拟网卡使用异步非阻塞

模型发送到内核。

4．论文按时完成的可能性

如果不改需求，不加需求，有充足性能的测试机，勉强可以。

5．参考文献

[1] RFC4301, Security Architecture for the Internet Protocol [S]. USA: S. Kent,K. Seo 2015.12

[2] RFC4303, IP Encapsulating Security Payload (ESP) [S]. USA: S. Kent. 2015.12

[3] RFC3602, The AES-CBC Cipher Algorithm and Its Use with IPsec [S]. USA: S. Frankel,R. Glenn,NIST,S. Kelly,Airespace 2003,9

[4] RFC2404, The Use of HMAC-SHA-1-96 within ESP and AH [S]. USA: C. Madson,Cisco Systems Inc,R. Glenn,NIST 1998,11

[5] RFC2367, PF_KEY Key Management API, Version 2 [S]. USA: D. McDonald,C. Metz,B. Phan 1998,7

[6] 阚闯,栾新,戚玮玮. 基于 Linux 的 XFRM 框架下 IPSec VPN 的研究[J]. 计算机工程. 2008(20)

[7] 孙云霄. 面向企业用户的高性能 VPN 系统的设计与实现[D]:[硕士毕业论文]. 威海: 哈尔滨工业大学(威海). 2015.

[8] 穆瑞超. 基于DPDK的高性能VPN网关的研究与实现[D]:[硕士毕业论文]. 威海: 哈尔滨工业大学(威海). 2017.

[9] 吴承. 用户态 IPSec 协议栈的研究与实现[D]:[硕士毕业论文]. 西安: 西安电子科技大学. 2014.

[10] 唐宏,柴桌原,任平,王勇. DPDK 应用基础[J]. 电信科学. 2016(09)

[11] 朱河清. 深入浅出 DPDK[M]. 机械工业出版社. 2016

[12] 王冠群. IPSec 中间人攻击检测方法与防御策略[D]:[硕士毕业论文]. 威海: 哈尔滨工业大学(威海). 2018.

[13] 廖悦欣. IPSec 协议实现技术研究[D]:[硕士论文]. 华南理工大学. 2013.

指导教师评语：_____

_____

_____

_____

_____

指导教师签字：_____　　检查日期：_____