Technology Guide

FD.io VPP-SSwan and Linux-CP - Integrate StrongSwan with World's First Open Sourced 1.89 Tb IPsec Solution

Authors

Roy Fan Zhang

Georgii Tkachuk

Pablo De Lara Guarch

Tomasz Kantecki

Kai Ji

John DiGiglio

1 Introduction

FD.io Vector Packet Processing (VPP) IPsec is an important component in VPP to enable secure, reliable, and fast networking applications. VPP IPsec provides a set of easy-to-use CLI and VAPI commands for users to configure Security Policy Database (SPD), Security Associations (SA), and associated cryptographic algorithms and keys.

With VPP IPsec running on a single 3rd Gen Intel® Xeon® Scalable processor core, one can achieve 31 Gbps throughput for a single Security Association for tunnel IPsec with AES-GCM-128 cryptography algorithm (IPSec IPv4 Routing, 2023[1]), over six times of what can be achieved with Linux Kernel based IPsec. For a 4th Gen Intel® Xeon® processor, the system performance can even achieve up to 1.89 Terabit No Drop Rate (NDR) IPsec tunnel throughput with 40 CPU cores in a single processor package, equivalent to almost 50 Gbps per CPU core[2]. With such a high throughput advantage, switching from kernel IPsec to Fd.io VPP IPsec appears as an obvious solution. However, it also brings new problems to be solved. IPsec relies on secured methods to setup SAs between two endpoints. The protocol to handle the SA setup is Internet Key Exchange (IKE). Fd.io VPP IPsec contains a mature, performant, and widely used IPsec implementation, but it is an incomplete IKEv2 implementation that is not production ready.

One of the most widely used IKE applications is StrongSwan, which is a multiplatform IPsec implementation. StrongSwan has complete IKE suites with a plugin architecture to handover IPsec data plane processing to either in-house or third-party implementations such as Linux/Windows kernel etc. This enabled integrating VPP IPsec into StrongSwan.

This guide demonstrates how to use VPP-SSwan and Linux Control Plane, two new plugins in VPP, to integrate VPP with StrongSwan seamlessly, so that the user can replace StrongSwan's Linux kernel IPsec data path to significantly improve IPsec performance from VPP.

This document is intended for communication service providers, or anyone looking to improve the IPsec throughput performance in their network. Even though the goal of this document is to showcase StrongSwan and IPsec VPP integration, the technologies enabled here can be used as a reference point for integrating VPP with other IKEv2 Control Plane interfaces.

This document is part of the [Network & Edge Platform]().

[1] https://s3-docs.fd.io/csit/master/report/vpp_performance_tests/packet_throughput_graphs/ipsec-3n-icx-xxv710.html#b-2t1c-ipsec-aes256gcm-ip4routing-base-scale-sw-avf

[2] https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-high-performance-ipsec-with-4th-gen-intel-xeon-scalable-processor-technology-guide

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	May 2023	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
AEAD	Authenticated encryption with associated data
ARP	Address Resolution Protocol
BPF	Berkeley Packet Filter
CLI	Command line interface
DCA	Direct Cache Access
DMA	Direct Memory Access
DPDK	Data Plane Development Kit
DUT	Device Under Test
ESP	Encapsulating Security Payload
FD.io	Fast Data Input/Output
GRE	Generic Routing Encapsulation
Intel AVX-512	Intel® Advanced Vector Extensions 512
Intel® I/OAT	Intel® I/O Acceleration Technology (Intel® I/OAT)
IKE	Internet Key Exchange
MSI-X	Extended Message Signaled Interrupts
NAT	Network Address Translation
ND	Neighbor Discovery
NDR	No Drop Rate
RSC	Receive Side Coalescing
SA	Security Associations
SCTP	Stream Control Transmission Protocol
SPD	Security Policy Database
SVM	Shared Virtual Memory
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPP	Vector Packet Processing

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® I/O Acceleration Technology	https://www.intel.com/content/www/us/en/wireless-network/accel-technology.html
Introducing the Intel® Data Streaming Accelerator (Intel® DSA)	https://01.org/blogs/2019/introducing-intel-data-streaming-accelerator
VPP Wiki	https://wiki.fd.io/view/VPP
VPP IPsec Wiki	https://wiki.fd.io/view/VPP/IPSec
VPP Host Stack Wiki	https://wiki.fd.io/view/VPP/HostStack
VPP/Configure VPP TAP Interfaces For Container Routing	https://wiki.fd.io/view/VPP/Configure_VPP_TAP_Interfaces_For_Container_Routing
Getting Started with the debug CLI	https://s3-docs.fd.io/vpp/23.06/cli-reference/gettingstarted/index.html
Fast Multi-buffer IPsec Implementations on Intel® Architecture Processors	https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/fast-multi-buffer-ipsec-implementations-ia-processors-paper.pdf
Intel® Multi-Buffer Crypto for IPsec GitHub	https://github.com/intel/intel-ipsec-mb

2 Overview

This document is intended as a guide to show how to integrate StrongSwan with Fd.io VPP, so that the user can benefit from both StrongSwan's friendly UI and VPP IPsec's performance.

This document includes the step-by-step guide to compile and install VPP and the VPP-SSwan plugin for StrongSwan and execute basic tests between two containers, one running StrongSwan + Linux Kernel IPsec and the other running StrongSwan + VPP IPsec. The tests includes IKEv2 handshake, SA negotiating, and ping tests between the containers.

2.1 Linux Kernel IPsec Implementation

Linux Kernel IPsec is currently one of the most widely used IPsec solutions. It has the reputation of a rich algorithm support list, actively maintained, and mostly bug free. However, the performance of Linux Kernel IPsec is not optimal; the IPSec throughput for a single CPU core with a single SA IPsec is no more than 5 Gbps (Cryptographic Acceleration, 2018[3]).

Intel® Multi-Buffer Crypto for IPSec is a family of highly optimized software implementations of the symmetric cryptographic algorithms. With the rich and easy-to-use APIs provided by Intel Multi-Buffer Crypto for IPSec, the user can easily make full use of the CPU's latest cryptographic accelerations provided by Intel including the new vAES and vPCLMUL instructions. These Intel® Advanced Vector Extensions 512 (Intel® AVX-512) accelerated instructions allow processing of up to four 128-bit AES blocks in parallel, getting theoretically four times better performance compared with 3rd Gen Intel Xeon Scalable processors. Moreover, the Intel Multi-Buffer Crypto library hides all implementation details to accommodate different CPU flags (SSE, AVX, AVX2, AVX512) behind the APIs, to ensure the highly optimized cryptographic operation results for all Intel® CPUs in the market and provides the user seamless transition of their code into new 3rd and 4th Gen Intel Xeon Scalable processor based systems.

2.2 VPP IPsec Kernel Bypass Implementation

2.2.1 Fast Data Input/Output (FD.io), Vector Packet Processing (VPP)

FD.io (Fast Data Input/Output) is a Linux Foundation Open Source Project that provides fast network packet processing capability. FD.io Vector Packet Processing (VPP) is one of the many sub-projects within FD.io that provides L2-L4 stack processing.

The term "vector" in VPP is essentially a group of packets, referred to as a "vector of packets" or a "packet vector". Each function block treats a packet vector (currently with a maximum of up to 256 packets) as input and processes them in the same manner. This helps to maximize the efficiency of the CPU instruction cache (I-cache). In addition, VPP innovatively adopts a Packet Processing Graph as its core design, where each function block is abstracted as a graph node. The graph nodes are organized into tree shaped graphs by registering the "next" output nodes either initially or during run-time. The packet vectors will be flowing from network adapter RX nodes all the way to TX nodes (or dropped) based on the destination processing nodes of each node.

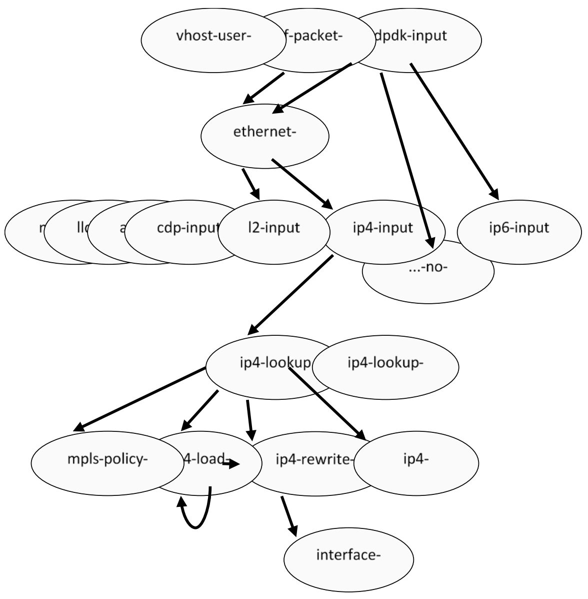


Figure 1. An example of a VPP Packet Processing Graph

[3] https://libreswan.org/wiki/Cryptographic_Acceleration

The packet processing graph has the distinctive advantage of being highly flexible; the new graph nodes can be "plugged in" anywhere and existing ones can be bypassed via simple software or real-time command line configuration. It is also efficient as the run-to-completion model ensures that the vector of the packets remain within the CPU data cache throughout the processing pipeline.

For more information on VPP, refer to the links provided in the References section of this document.

2.2.2 VPP IPsec

VPP IPsec is an important component in VPP towards providing a secure, reliable, and fast networking application. VPP IPsec provides a set of easy-to-use CLI and VAPI commands for user to configure the Security Policy Database , the Security Associations and the associated cryptographic algorithms and keys. Please refer to the VPP IPsec Wiki page for full cryptographic functionality support.

The most resource consuming procedure within IPsec is the symmetric cryptographic operation. To ensure both the performance and the flexibility of cryptographic operation, VPP IPsec takes advantage of the underlying crypto infrastructure.

With VPP IPsec running on a single 4th Gen Intel Xeon Scalable Processor core, one can achieve up to 50 Gbps of throughput for a single SA with a tunnel based IPSec with the AES-GCM-128 algorithm, over six times what Linux kernel based IPSec can achieve. A computer system with same processor can achieve up to 1.89 Tera-bit No Drop Rate IPsec tunnel throughput with 40 CPU cores.

2.3 IKEv2 and StrongSwan

With such a high throughput advantage, switching from kernel IPsec to Fd.io VPP IPsec seems to be a must. However, the actual implementation is more complex. IPsec relies on a secured method to setup SAs between two endpoints. The protocol to handle the SA setup is Internet Key Exchange (IKE). IKE is the protocol to set up a shared session secret between the endpoints, which will be used to derive the symmetric cryptography key securely. IKE is currently at version 2 (IKEv2). Compared to IKE, IKEv2 has the advantages of NAT traversal, SCTP support, DoS attach resilience, and much more. Fd.io VPP IPsec contains a mature , performant, and widely used IPsec implementation, however it is an incomplete IKEv2 implementation that is not production ready. To make VPP IPsec widely adopted we need an alternative IKEv2 backend. Adding support for a popular IKE application to VPP also helps in simplifying the user migration to VPP IPsec solution.

One of the most widely used IKE applications is StrongSwan[4]. StrongSwan has complete IKE suites with a plugin architecture to handover IPsec data plane processing to either in-house or third-party implementations such as Linux/Windows kernel, etc. This enabled integrating VPP IPsec into StrongSwan.

2.4 Connecting Kernel with Userspace I/O - Previous Solutions

Even though VPP IPsec is integrated with StrongSwan, the user still cannot benefit from StrongSwan's convenient IKE implementation and VPP's highly performant IPsec implementation directly. StrongSwan installs the routes into kernel routing tables. StrongSwan expects that the kernel diverts the IKE traffic to it and processes the IPsec data path traffic (encrypt and encapsulate a plain IPv4/v6 packet, or decrypt and decapsulate an encrypted ESP packet), both outbound and inbound, by itself. For a single endpoint, both the IKE traffic and IPsec data flow will be received by the same network adapter. This requirement is limited by the long-term Linux kernel and kernel-bypass problem: the kernel-bypass method tends to have the user space take over the network adapter, bypassing the kernel completely. This means, to make VPP and StrongSwan work together, one must decide whether the VPP or the Linux kernel driver will receive the network packets from the network adapter and configure both systems to intelligently route the expected traffic between kernel and user space, and if needed, transmit the processed packets to a second network adapter.

To build such connection between kernel and user space I/O, there were two possible solutions, each with its own disadvantages:

- Using the VPP host interface (with Linux af_packet interface as a pair)
- Using the VPP AF_XDP interface (with Linux XDP socket as a pair)

2.4.1 Connect Kernel and User Space I/O with af_packet

Consider the kernel network adapter as one example. To offload the encrypted ESP traffic to VPP, the user must create a host interface (Linux af_packet interface) to share with the Linux kernel and configure from the Linux kernel side to route the expected traffic to VPP through the corresponding interface. The kernel may expect the decrypted plain packets back from VPP hence, the user has to configure the routing properly on the VPP side. The situation is similar when the Linux kernel offloads clear

[4] https://docs.strongswan.org/docs/5.9/howtos/introduction.html

text traffic to be encrypted by VPP. Configuring such complex routing for both the Linux kernel and VPP from StrongSwan through simple swanctl command is challenging because:

1. StrongSwan must be updated to understand VPP af_packet interfaces.
2. A single routing entry needs to be configured for both kernel and VPP separately.
3. Even if everything is working properly, the overall IPsec throughput will be limited by slow af_packet interfaces.

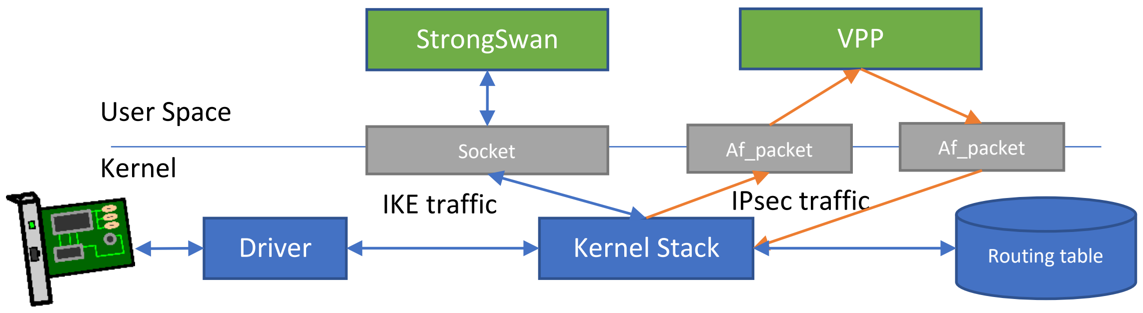


Figure 2. VPP and StrongSwan with a Kernel Network Adapter

2.4.2 Connect Kernel and User Space I/O with af_xdp

An alternative solution with the kernel network adapter is to use AF_XDP with a custom BPF filter program. AF_XDP is a socket type for fast raw packet processing. AF_XDP allows the network packet to bypass the Linux kernel stack and pass from the network adapter kernel driver directly to user-space applications. AF_XDP can achieve 11.5 Mpps with 0% packet loss based on a simple L2fwd micro benchmark shipped with the Linux kernel[5].

In addition to passing all network packets to user space, it is possible to define static rules to pass only select network flows through a static eBPF program. For example, you can write an eBPF program to pass only IP packets with an ESP header to user space through the XDP socket.

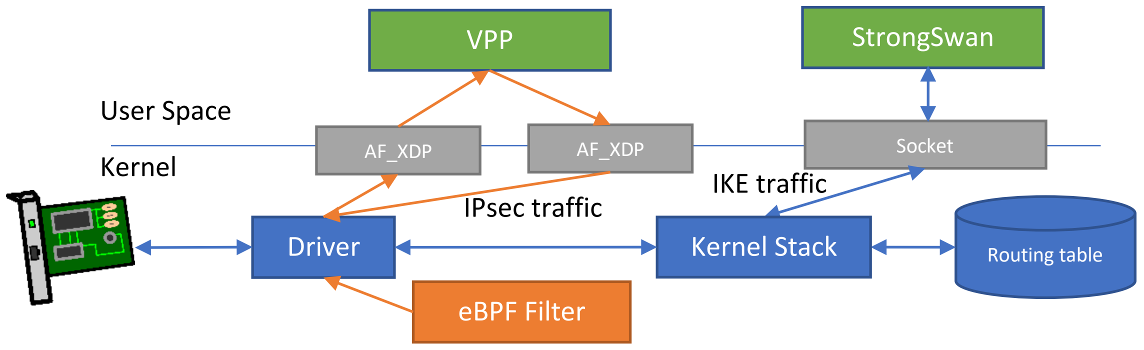


Figure 3. VPP and StrongSwan with AF_XDP

The solution, although performant, has its limitations as well:

1. AF_XDP requires a relatively newer Linux kernel (first released in Linux 4.18) and depends on the network adapter kernel driver support. Some advanced features such as a custom BPF program loads relies on even newer kernel versions and updated network adapters. Without the updated Linux kernel and network adapters that support the feature required, the above solution will not work.
2. An eBPF filter program is static and hard to scale. When a new IPsec SA is created, or an existing IPsec SA is updated/removed, the filter program needs to be updated to include the new routing rules and the updated byte code is to be generated accordingly. In addition, connecting the new filter program to the network adapter may interrupt the traffic.

[5] https://networkbuilders.intel.com/solutionslibrary/af-xdp-sockets-high-performance-networking-for-cloud-native-networking-technology-guide

3 Our Solution: VPP-SSwan with Linux Control Plane

3.1 VPP-SSwan

VPP-SSwan is a StrongSwan plugin that offloads the IPsec outbound and inbound data plane packet processing to Fd.io VPP. The plugin takes advantages of the VPP C API[6] that translates the IPsec SA add and delete requests and routing rules into VPP socket messages. VPP-SSwan has the following advantages:

1. Easy to use: VPP-SSwan was written following the StrongSwan Charon specifications and provides helpful scripts to compile and install the plugin. The user only needs to run single commands or manually copy the compiled VPP-SSwan plugin and configuration file to specific StrongSwan locations, given StrongSwan was installed in a custom location in the system before restarting StrongSwan service.
2. Performant: VPP-SSwan takes advantage of the highly optimized VPP I/O and IPsec stack. This means the network I/O and IPsec data path are processed by VPP.

The VPP-SSwan plugin is included in VPP from 22.10 release and works with StrongSwan 5.9.5 onwards.

VPP-SSwan expects VPP to take over the network I/O and StrongSwan expects the control path (IKE) packets and data path (ESP) packets sharing the same IP addresses. VPP provides a way to route the incoming IKE packets to the Linux kernel stack and the outgoing IKE packets to the protected network adapters, preferably without separate routing configurations for them. This is where the VPP Linux Control Plane plugin becomes useful.

3.2 Linux Control Plane Plugin

Linux Control Plane[7], or Linux-CP in short, is a VPP plugin that allows VPP to integrate into the Linux kernel. The idea behind Linux-CP is to provide a mirrored interface pair between the VPP owned network adapter and a Tun/Tap interface linked between VPP and the Linux kernel. The network traffic is cross connected between the Linux Tun/Tap interface and the paired VPP network adapter.

Linux-CP configures the routing between the Tun/Tap interface and the network adapter and syncs the ARP/ND table between VPP and Linux automatically. When the network adapter is added as an IP address through VPP, Linux-CP will capture the command and automatically configure the Tun/Tap interface with the same IP address in the Linux kernel side and ensures that the packets with the corresponding IP address are routed between network adapters and the Tun/Tap interface within VPP. In addition, Linux-CP makes sure it is the Linux kernel that owns the [ARP/ND] neighbor tables, although the entries are copied to VPP.

3.3 How VPP-SSwan and Linux-CP Function Together

Figure 4 shows an example of how IKE messages are flowing between a VPP owned network adapter and StrongSwan with the help of Linux-CP. In this example, we assume StrongSwan expects the IKE messages and encrypted ESP packets will have the local IP address of 10.0.0.1. To route the IKE packets from the network adapter owned by VPP to StrongSWAN, we create a Linux-CP instance, which binds the mirrored interface pair (WAN network adapter port and Tun/Tap port in the figure), both with the same IP address 10.0.0.1. Linux-CP helps to configure the routing between the interfaces automatically. When an IKE message is received by VPP through the WAN network adapter port, it will be routed to the kernel through the Tun/Tap port and is processed by the Linux kernel stack before being passed to StrongSwan. If the SA is negotiated successfully, StrongSwan will configure the IPsec ESP flow and routing in VPP through the VPP-SSwan plugin. Once the automated configuration is finished, VPP will take care of IPsec ESP encryption/decryption as well as forwarding the packets to the desired ports.

Assume two subnets 192.168.0.0/24 and 192.168.1.0/24 connected via the Internet want to set up a secured IPsec tunnel. Each subnet has an IPsec gateway system with WAN IPs of 10.0.0.1 and 10.0.0.2 respectively. The StrongSwan applications on each gateway system negotiated a common SA and privately created a session symmetric cryptography algorithm and key. The StrongSwan application that offloads the data plane processing to VPP will then use VPP's CAPI to create the Security Policy (SP) and SA. VPP-SSwan will make sure the necessary SA/SP creation is parsed correctly to VPP CAPI calls. Linux-CP will also deal with the correct routing configuration between the WAN network adapter and the mirrored Tun/Tap port.

[6] https://wiki.fd.io/view/VPP/How_To_Use_The_C_API

[7] https://s3-docs.fd.io/vpp/22.06/developer/plugins/lcp.html

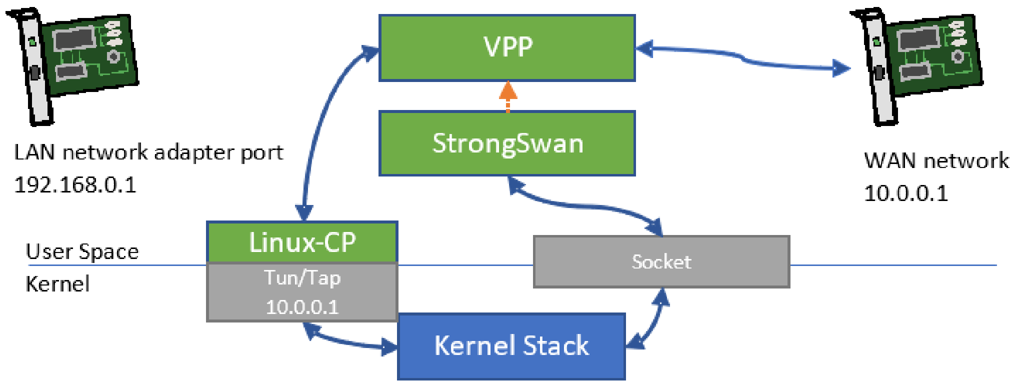


Figure 4. IKE Traffic Between VPP Owned Network Adapter and StrongSwan Application

From this point, since VPP owns both the LAN and WAN network adapters, it will receive the packets from them and process routing and IPsec outbound and inbound operation accordingly. When network packets are received by the LAN network adapter and match an outbound security policy (SP) rule, they undergo encryption. This process involves adding an ESP (Encapsulating Security Payload) header to the packets. Once encrypted, the packets are forwarded to the WAN network adapter port.

On the other hand, when network packets are received by the WAN network adapter, they are checked against any applicable inbound SP rule. If a rule is matched, the packets are decrypted, and the ESP header or tunnel header is removed. Subsequently, the decrypted packets are forwarded either to the LAN network adapter or to the Tun/Tap port, depending on the IP destination address found in the decrypted packets. VPP-SSWAN will also make sure that the IPsec bypass/drop rules are handled accordingly.

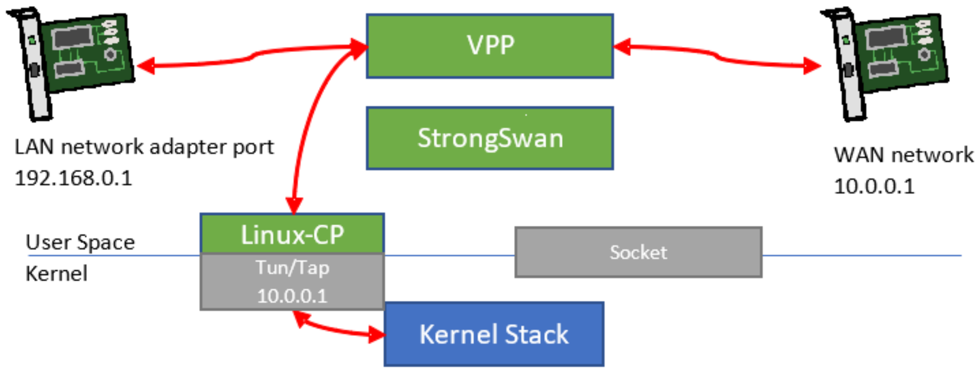


Figure 5. IPsec Traffic Between VPP Owned Network Adapter and Kernel Owned Tun/Tap Port

3.5 How to Use VPP-SSwan and Linux-CP to Integrate VPP with StrongSwan

VPP-SSwan has been integrated in mainstream VPP since VPP 22.10 release. The location of VPP-SSwan is in vpp/extras/strongswan/vpp_sswan. The solution provides some handy automated scripts to help compile and install the plugin easily.

3.5.1 Download and Compile VPP

To get started developing with VPP, you need to get the required VPP sources and then build the packages. For VPP build guide and system prerequisites, refer to the VPP getting started guide:

https://s3-docs.fd.io/vpp/23.02/gettingstarted/installing/index.html

3.5.2 Build VPP-SSwan Plugin

VPP-SSWAN requires StrongSwan source to compile. To build VPP-SSWAN, use this command:

```
cd <VPP directory>/extras/strongswan/vpp_sswan/
make all
```

3.5.3 Install StrongSwan 5.9.5 or Later (optional)

It is recommended to use StrongSwan version 5.9.6 or 5.9.5 from VPP-SSWAN to be compiled and integrated.

1. Download StrongSwan source code to:

```
cd <vpp directory>/build/external/downloads
```

```
wget https://github.com/strongswan/strongswan/archive/refs/tags/5.9.5.zip
```

2. Unzip StrongSwan source code to

```
<vpp directory> /build-root/build-vpp-native/external/sswan
```

3. Configure strongswan by using this command:

```
cd <vpp directory> /build-root/build-vpp-native/external/sswan/strongswan-5.9.5
```

```
./autogen.sh
```

```
./configure --prefix=/usr --sysconfdir=/etc --enable-libipsec --enable-systemd --enable-swanctl --
disable-gmp --enable-openssl
```

```
make
```

```
make install
```

4. Alterative you can pull StrongSwan source zip file and make install in vpp-sswan directory:

```
cd <VPP directory>/extras/strongswan/vpp_sswan/
```

```
make pull-swan
```

```
make install-swan
```

3.5.4 Install VPP-SSwan Plugin into StrongSwan

After the VPP-SSWAN plugin has been built and StrongSwan was installed on the host, the following commands will install VPP-SSWAN plugin into StrongSwan

```
cd <VPP directory>/extras/strongswan/vpp_sswan/
```

```
make install
```

Or you can manually copy libstrongswan-kernel-vpp.so into /usr/bin/ipsec/plugins, and kernel-vpp.conf into /etc/strongswan.d/chron/

The StrongSwan service needs to be restarted to apply the newly changes:

```
systemctl restart strongswan.service
```

3.5.5 Restart StrongSwan

As an example, swanctl.conf file provides an example configuration to initialize connections between two endpoints.

You may update the file based on your need and copy into:

/etc/swanctl/conf.d/swanctl.conf

3.5.6 Buildup VPP-SSwan Docker Images

Starting from the VPP 22.10 release, the vpp-sswan Docker test scripts now offer an integrated testing environment for creating Docker images, bringing up VPP, setting up network interfaces, and deploying StrongSwan configurations. The test scripts include a run script that builds two container images, with VPP, DPDK and StrongSwan installed. These pre-defined images use a swanctl.conf configuration file to establish a Security Association (SA) between the two containers.

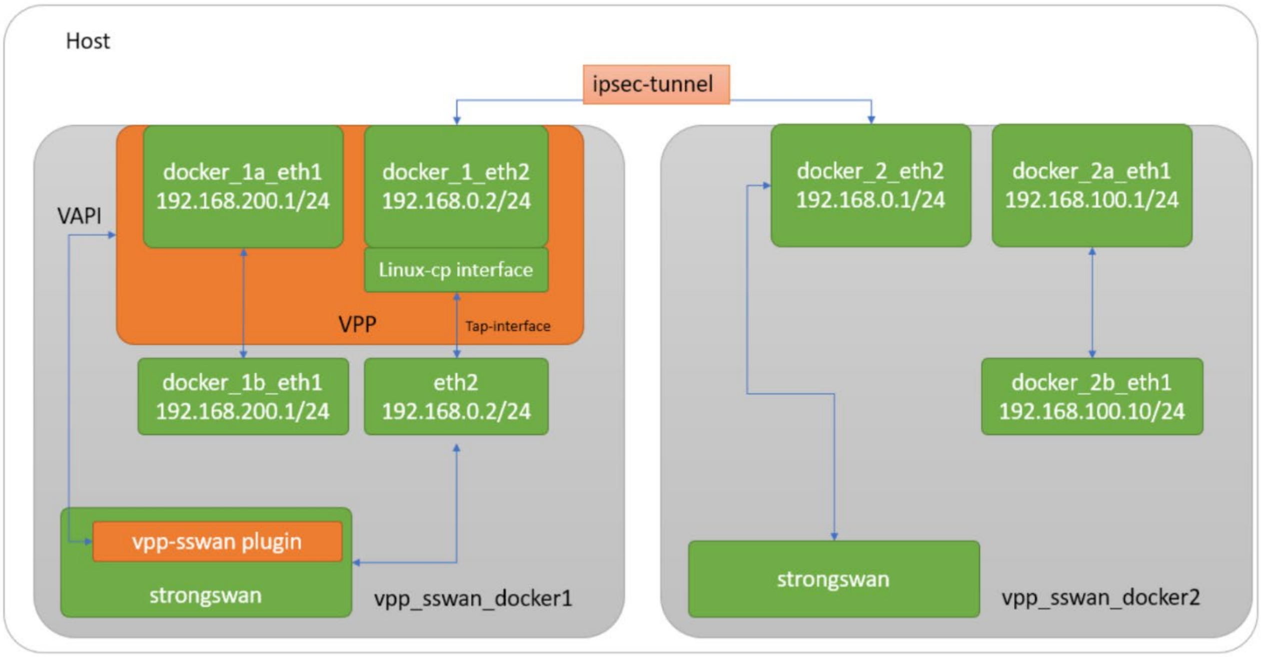
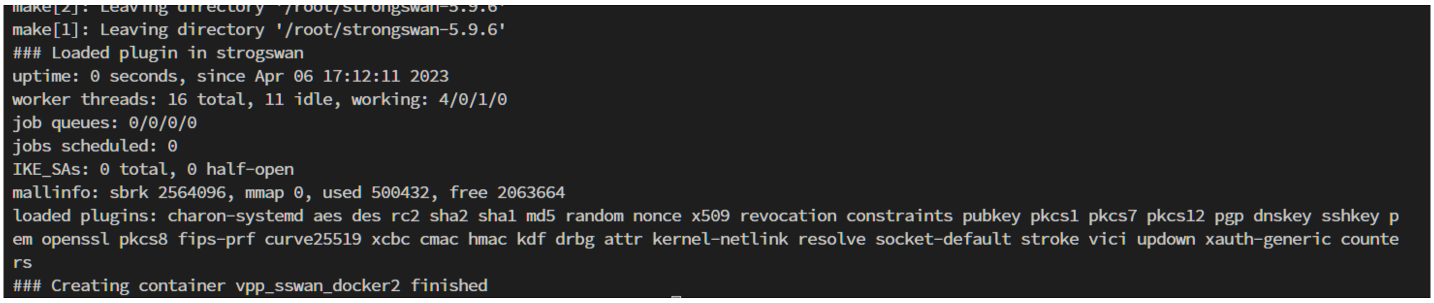

Host
ipsec-tunnel
VAPI
docker_1a_eth1 192.168.200.1/24
docker_1_eth2 192.168.0.2/24
Linux-cp interface
VPP
Tap-interface
docker_1b_eth1 192.168.200.1/24
eth2 192.168.0.2/24
vpp-sswan plugin
strongswan
vpp_sswan_docker1
docker_2_eth2 192.168.0.1/24
docker_2a_eth1 192.168.100.1/24
docker_2b_eth1 192.168.100.10/24
strongswan
vpp_sswan_docker2

Figure 6. VPP SSwan Docker Testing Network Topology

3.5.6.1 Create and Prepare Containers

The run.sh scripts will automatically download vpp and dpdk source code, compile, and install into the Docker images.

```
sudo ./extras/strongswan/vpp_sswan/docker/run.sh prepare_containers
```



```
make[2]: Leaving directory '/root/strongswan-5.9.6'
make[1]: Leaving directory '/root/strongswan-5.9.6'
### Loaded plugin in strogswan
uptime: 0 seconds, since Apr 06 17:12:11 2023
worker threads: 16 total, 11 idle, working: 4/0/1/0
job queues: 0/0/0/0
jobs scheduled: 0
IKE_SAs: 0 total, 0 half-open
mallinfo: sbrk 2564096, mmap 0, used 500432, free 2063664
loaded plugins: charon-systemd aes des rc2 sha2 sha1 md5 random nonce x509 revocation constraints pubkey pkcs1 pkcs7 pkcs12 pgp dnskey sshkey p
em openssl pkcs8 fips-prf curve25519 xcbc cmac hmac kdf drbg attr kernel-netlink resolve socket-default stroke vici updown xauth-generic counte
rs
### Creating container vpp_sswan_docker2 finished
```

3.5.6.2 Apply VPP_SSwan Configuration in Docker Containers

The config command configures all virtual pair network adapters in kernel , initializes and establishes connection in Strongswan

```
sudo ./extras/strongswan/vpp_sswan/docker/run.sh config
```

```
### Configuration vpp_sswan_docker1 and vpp_sswan_docker2
### Adding network namespace for vpp_sswan_docker1 and vpp_sswan_docker2
netns of vpp_sswan_docker1 exposed as /var/run/netns/vpp_sswan_docker1
netns of vpp_sswan_docker2 exposed as /var/run/netns/vpp_sswan_docker2
### Adding network namespace for vpp_sswan_docker1 and vpp_sswan_docker2 finished
### Setting network for vpp_sswan_docker1 and vpp_sswan_docker2
### Setting network for vpp_sswan_docker1 and vpp_sswan_docker2 finished
### Running VPP and sswan on: vpp_sswan_docker1 and vpp_sswan_docker2
### Checking connections between VPP and Strongswan
Shared memory clients
                Name      PID   Queue Length           Queue VA Health
          strongswan    79938              0 0x00000001301ce800 OK
### Running VPP and sswan on: vpp_sswan_docker1 and vpp_sswan_docker2 finished
### initiate SSWAN between vpp_sswan_docker1 and vpp_sswan_docker2
[IKE] initiating IKE_SA net-net[1] to 192.168.0.1
[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG) N(REDIR_SUP) ]
[NET] sending packet: from 192.168.0.2[500] to 192.168.0.1[500] (240 bytes)
[NET] received packet: from 192.168.0.1[500] to 192.168.0.2[500] (248 bytes)
[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG) N(CHDLESS_SUP) N(MULT_AUTH) ]
[CFG] selected proposal: IKE:AES_CBC_128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/CURVE_25519
[IKE] authentication of 'sun.strongswan.org' (myself) with pre-shared key
[IKE] establishing CHILD_SA net-net{1}
[ENC] generating IKE_AUTH request 1 [ IDi N(INIT_CONTACT) IDr AUTH SA TSi TSr N(MOBIKE_SUP) N(NO_ADD_ADDR) N(MULT_AUTH) N(EAP_ONLY) N(MSG_ID_SY
N_SUP) ]
[NET] sending packet: from 192.168.0.2[4500] to 192.168.0.1[4500] (304 bytes)
[NET] received packet: from 192.168.0.1[4500] to 192.168.0.2[4500] (272 bytes)
[ENC] parsed IKE_AUTH response 1 [ IDr AUTH SA TSi TSr N(MOBIKE_SUP) N(ADD_4_ADDR) N(ADD_4_ADDR) N(ADD_4_ADDR) ]
[IKE] authentication of 'moon.strongswan.org' with pre-shared key successful
[IKE] IKE_SA net-net[1] established between 192.168.0.2[sun.strongswan.org]...192.168.0.1[moon.strongswan.org]
[IKE] scheduling rekeying in 13313s
[IKE] maximum IKE_SA lifetime 14753s
[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA1_96/NO_EXT_SEQ
[KNL] policy have interface 192.168.0.2
[KNL] firstly created, spd for host-docker_1_eth2 found sw_if_index is 1
[KNL] policy FWD interface
[KNL] policy have interface 192.168.0.2
[IKE] CHILD_SA net-net{1} established with SPIs c57320b2_i c3729241_o and TS 192.168.200.0/24 === 192.168.100.0/24
[IKE] peer supports MOBIKE
initiate completed successfully
### initiate SSWAN between vpp_sswan_docker1 and vpp_sswan_docker2 finished
```

3.5.6.3 VPP_SSwan Validation

Once the Docker image is prepared correctly, the virtual Ethernet pairs should be connected to the different network namespace.

`ip netns exec vpp_sswan_docker1 ip a s`

```
 32: docker_1_eth2@if31: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
     link/ether d6:32:f9:1a:be:d8 brd ff:ff:ff:ff:ff:ff link-netns vpp_sswan_docker2
 33: docker_1b_eth1@docker_1a_eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
     link/ether 9e:f8:7f:17:bb:9f brd ff:ff:ff:ff:ff:ff
     inet 192.168.200.10/24 scope global docker_1b_eth1
        valid_lft forever preferred_lft forever
 34: docker_1a_eth1@docker_1b_eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
     link/ether 12:63:f1:26:f8:90 brd ff:ff:ff:ff:ff:ff
```

`ip netns exec vpp_sswan_docker2 ip a s`

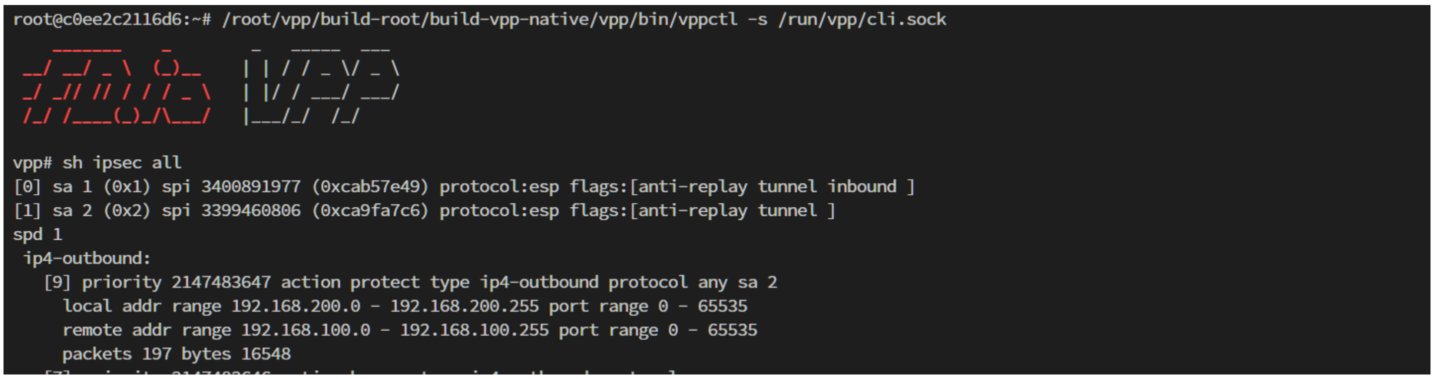
```
31: docker_2_eth2@if32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 92:05:0b:8e:f7:0a brd ff:ff:ff:ff:ff:ff link-netns vpp_sswan_docker1
    inet 192.168.0.1/24 scope global docker_2_eth2
       valid_lft forever preferred_lft forever
35: docker_2b_eth1@docker_2a_eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 8e:16:19:e0:56:bd brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.10/24 scope global docker_2b_eth1
       valid_lft forever preferred_lft forever
36: docker_2a_eth1@docker_2b_eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 26:3a:cb:25:cc:40 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 scope global docker_2a_eth1
       valid_lft forever preferred_lft forever
```

To enter the bash console inside of containers, use this command:

`sudo docker exec -it vpp_sswan_docker1 bash`

Show ipsec configuration in VPP inside of container:

```
/root/vpp/build-root/build-vpp-native/vpp/bin/vppctl -s /run/vpp/cli.sock
```

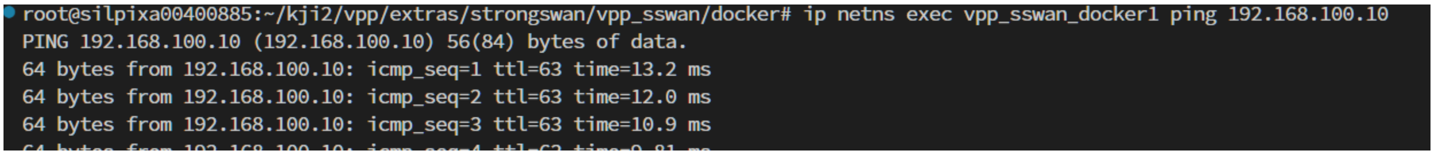


```
root@c0ee2c2116d6:~# /root/vpp/build-root/build-vpp-native/vpp/bin/vppctl -s /run/vpp/cli.sock
    _______    _        _   _____  ___
 __/ __/ _ \  (_)__    | | / / _ \/ _ \
 _/ _// // / / / _ \   | |/ / ___/ ___/
 /_/ /____(_)_/\___/   |___/_/  /_/

vpp# sh ipsec all
[0] sa 1 (0x1) spi 3400891977 (0xcab57e49) protocol:esp flags:[anti-replay tunnel inbound ]
[1] sa 2 (0x2) spi 3399460806 (0xca9fa7c6) protocol:esp flags:[anti-replay tunnel ]
spd 1
 ip4-outbound:
   [9] priority 2147483647 action protect type ip4-outbound protocol any sa 2
     local addr range 192.168.200.0 - 192.168.200.255 port range 0 - 65535
     remote addr range 192.168.100.0 - 192.168.100.255 port range 0 - 65535
     packets 197 bytes 16548
```

To send ARP messages from local (vpp_sswan_docker1) to remote (vpp_sswan_docker2), use this commend:

```
ip netns exec vpp_sswan_docker1 ping 192.168.100.10
```

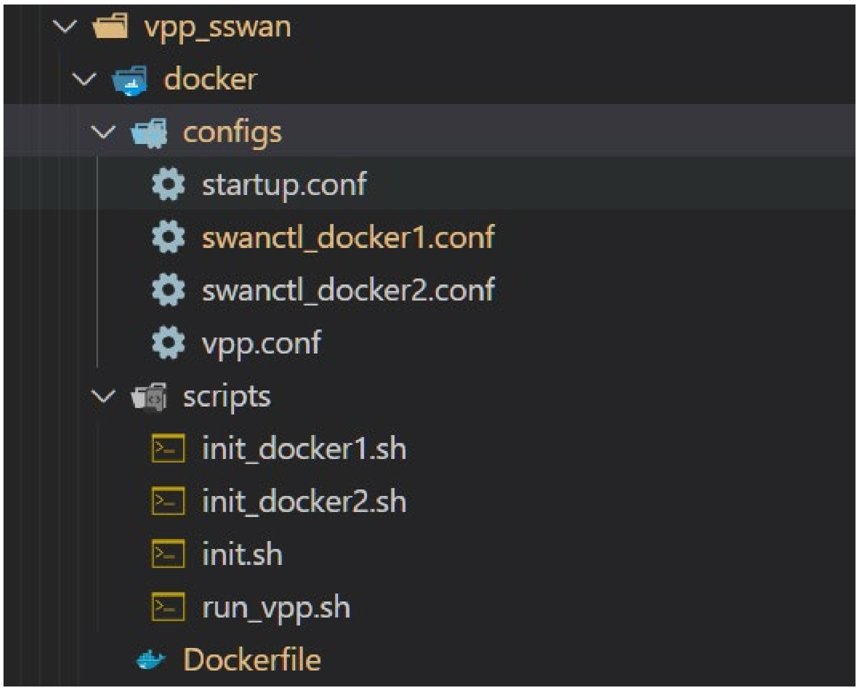


```
root@silpixa00400885:~/kji2/vpp/extras/strongswan/vpp_sswan/docker# ip netns exec vpp_sswan_docker1 ping 192.168.100.10
PING 192.168.100.10 (192.168.100.10) 56(84) bytes of data.
64 bytes from 192.168.100.10: icmp_seq=1 ttl=63 time=13.2 ms
64 bytes from 192.168.100.10: icmp_seq=2 ttl=63 time=12.0 ms
64 bytes from 192.168.100.10: icmp_seq=3 ttl=63 time=10.9 ms
```

The output of "ip4-outbound" packets should be updated accordingly based on the number of egress ping messages.

3.5.6.4 Configuration Files

All the configuration files used in Docker images and StrongSwan configuration with SA are available in "<VPP directory>/strongswan/vpp_sswan/docker" directory from VPP release 22.10.



```
vpp_sswan
  docker
    configs
      startup.conf
      swanctl_docker1.conf
      swanctl_docker2.conf
      vpp.conf
    scripts
      init_docker1.sh
      init_docker2.sh
      init.sh
      run_vpp.sh
    Dockerfile
```

4 Summary

VPP-SSwan is a software solution that combines the advantages of two popular open-source projects, StrongSwan and FD.io VPP (Vector Packet Processing). StrongSwan provides a user-friendly interface for setting up secure communication channels using VPNs (Virtual Private Networks), while VPP is a high-performance networking stack that uses hardware acceleration to achieve incredibly fast packet processing speeds on Intel® Xeon® platforms.

By merging these two technologies, VPP-SSwan provides users with an easy-to-use VPN solution that also delivers exceptional performance. With VPP's packet processing capabilities, VPP-SSwan can handle large volumes of network traffic with low latency, making it ideal for use in high-performance computing environments.

Furthermore, VPP-SSwan supports a wide range of protocols and encryption algorithms, offering users the flexibility to select the best option for their specific needs. With StrongSwan's user-friendly interface, setting up and managing VPN connections is simple, even for those with limited networking experience.

Overall, VPP-SSwan is an excellent choice for anyone looking for a VPN solution that combines ease-of-use with high-performance networking capabilities.

0523/DN/WIT/PDF 779538-001US