Pushing OpenVPN down the stack: Data Channel Offload (DCO)

Antonio Quartulli
OpenVPN Inc.
Pleasanton CA, USA
antonio@openvpn.net

Abstract

OpenVPN is a userspace software responsible for creating an encrypted tunnel between two peers (peer-to-peer mode) or a central server and multiple clients (peer-to-multipeer mode). Until now both the control and the data planes were implemented in userspace, leading to notable performance penalty. The technique described in this paper, known as data channel offloading, consists in moving the data plane (i.e. user payload processing) to kernel space in order to reduce context-switching and thus improve the measurable tunnel throughput.

Keywords

VPN, tunnel, crypto, networking, device driver, netlink, acceleration, offload

Introduction

OpenVPN Data Channel Offload (ovpn-dco for brevity) is a Linux kernel module that implements the OpenVPN data plane (i.e. the fast path). By keeping user payloads in kernelspace, unneeded context-switches and expensive data copies are avoided, thus increasing performance, which results in a notable throughput improvement for the user.

This novel kernel module has been developed with simplicity in mind, therefore only a small subset of the OpenVPN features was selected for being implemented in ovpn-dco. Features of dubious gain and legacy knobs that are not meaningful anymore (i.e. compression) will not be ported to kernel space.

Architecture

The OpenVPN software in userspace is still the main actor as it is responsible for establishing a connection with a peer/server and perform:

- user authentication
- X509 certificates exchange
- parameters negotiation
- session key derivation

All the functions mentioned above happen on the control plane and it is important to point out that none of those is implemented in kernelspace. This is key to ovpn-dco as it shows how non-performance-critical tasks are kept in userspace, thus limiting the kernel code complexity and attack surface.

After a connection has been established, userspace informs ovpn-dco about the new peer. Along with various attributes, also the socket used to communicate with the other entity is passed to ovpn-dco. From this moment on, userspace is expected to not use this socket anymore when sending/receiving control packets to/from this peer.

Further communication on the control plane (like periodic key renegotiations) has to happen using the provided Netlink API, as described in the related section about userspace APIs.

All kernel-to-userspace messages are exchanged by means of GENL (Netlink) API.
Creating and destroying an interface is instead performed via RTNL.

Packet processing

Encapsulation and encryption

OpenVPN is known for supporting a variety of ciphers (as provided by the system SSL library), however, with the goal of keeping ovpn-dco as slim as possible, only AES-GCM and ChaCha20Poly1305 are currently supported by the kernel module implementation. Support for other AEAD ciphers may be easily added.

After the protocol handshake is completed and both peers have generated a session key, the latter is communicated by the userspace software, along with the selected algorithm and a nonce, to ovpn-dco. The whole state required to perform encryption and decryption operations is then generated and stored in kernel.

Thanks to the flexibility provided by the internal ovpn-dco data structures, each peer can be potentially configured with a different cipher, thus allowing userspace to pick the best among the supported ones.

TX and RX paths

Encryption and decryption happens respectively in the TX and RX traffic path. For each direction a ptr-ring is used to store packets waiting to be handled, while a worker picks one at a time and processes them.

During transmission packets are encrypted and then immediately forwarded to the destination peer using the udp-tunnel

APIs.

On the other hand, received packets are first decrypted and then queued for delivery to the networking interface via NAPI.

Routing

The OpenVPN userspace software can function in two modes, and so does ovpn-dco: peer-to-peer or multi-peer.
Peer-to-peer mode is activated when a host wants to connect to a server (in this case the host is also called 'client') or when it wants to connect to another peer-to-peer host.
In this configuration ovpn-dco accepts only one peer and any attempt to install a second entry simply results in substituting the previously existing one.
This setup is the simplest from the routing perspective, because there is no routing at all: any packet sent to the networking interface is encapsulated/encrypted and sent to the configured peer regardless of the actual destination. It doesn't matter if on the other side there is another peer or a server.

Multi-peer mode is instead activated when OpenVPN runs as a server. Under this configuration ovpn-dco can accept up to 4096 peers, as long as they have a unique ID and VPN IP (both v4 and v6).
In this case, upon a packet entering the networking interface, a destination peer must be selected among the available ones. To do so we have two possibilities:

1. the packet destination IP matches the VPN IP of a known peer: the latter is selected as destination;
2. the packet destination IP does not match the VPN IP of any known peer: the system routing table is searched for a route including the destination IP;
   - if no route is found, the packet is dropped (no route to host);
   - if a route is found, the nexthop is retrieved and is compared to the VPN IP of all known peers: the matching peer is selected as destination.

The approach explaned at point 2 above allows ovpn-dco to rely on the system routing table and thus avoid implementing a private routing mechanism. At the same time, users can inform ovpn-dco about new routes by simply adding them to the system routing table, as usual.

To configure ovpn-dco in peer-to-peer or multipeer mode, a specific flag must be passed along with the RTNL RTM_NETLINK command, namely IFLA_OVPN_MODE, which values can be either OVPN_MODE_P2P or OVPN_MODE_MP. The mode is then unalterable throughout the interface lifecycle.

Userspace API

To manage an ovpn-dco networking device, there are currently two sets of APIs that need to be used:

1. RTNL: to create and destroy interfaces of type 'ovpn-dco';
2. GENL (new 'ovpn-dco' family): to manage the OpenVPN specific aspects.

While the RTNL API is well known and used by many (all?) device drivers, the GENL API is specific to ovpn-dco and thus it's implemented in its code base.

Please note that the GENL API is not definitive and it may still change.

GENL: ovpn-dco

GENL messages assume that an ovpn-dco interface was already created and expect its ifindex to be always specified.

Available GENL userspace to kernelspace messages are:

- OVPN_CMD_NEW_PEER: inform about a new peer
- OVPN_CMD_GET_PEER: retrieve data about existing peer(s)
- OVPN_CMD_SET_PEER: set attributes on an existing peer (i.e. keepalive timeout)
- OVPN_CMD_DEL_PEER: delete an existing peer
- OVPN_CMD_NEW_KEY: configure a new key (and cipher) for a peer
- OVPN_CMD_SWAP_KEYS: swap keys stored in the primary and secondary slots
- OVPN_CMD_DEL_KEY: delete a an existing key
- OVPN_CMD_REGISTER_PACKET: register a userspace listener process for receiving control packets
- OVPN_CMD_PACKET: send a control packet over the wire

Among the above, the following can also be sent from kernelspace to userspace:

- OVPN_CMD_DEL_PEER: inform that a peer was deleted, with reason (multicast)
- OVPN_CMD_PACKET: send a non data-packet to the registered listener process (unicast)

As previously mentioned, it should be noted that after creating a new peer by means of OVPN_CMD_NEW_PEER, the socket passed down to the kernel should not be used anymore by userspace. For this reason the OVPN_CMD_REGISTER_PACKET and OVPN_CMD_PACKET messages are required to exchange control packets with the peers.

This said, it may be possible to simplify this model and let userspace still send/receive control packets using the socket. This topic is open for discussion/suggestions.

GENL: documenting attributes

A novel approach, open for discussion, has been used to declare all attributes being sent along with GENL messages. As it is possible to see in the ovpn_dco.h uapi header file (available at https://github.com/OpenVPN/ovpn-dco/blob/master/include/uapi/linux/ovpn_dco.h), attributes are not mixed in a big enum, but rather grouped by message that they are supposed to be attached to.

With this approach I strive to clearly document what attributes are expected by each message, without requiring a developer to dig into the ovpn-dco netlink implementation.

Userspace integrations

In order to use ovpn-dco, userspace software had to be modified to accommodate the new interface type and its management plane.

As of now there are two software implementations that integrate with ovpn-dco:

1. OpenVPN3-Linux client:
   https://github.com/OpenVPN/openvpn3-linux
2. OpenVPN2, master branch (soon to be v2.6):
   https://github.com/OpenVPN/openvpn

Both are publicly available for evaluation and testing.

The (out-of-tree) ovpn-dco code can be retrieved at https://github.com/OpenVPN/ovpn-dco .

An OpenWrt feed is also available at https://github.com/OpenVPN/openvpn-dev-openwrt .
In this feed it is possible to find OpenVPN and ovpn-dco both being built out of their respective master/main branch.

Performance

In this section I present results obtained by running iperf speed tests between two Ubuntu 22.04 virtual machines running on a host equipped with an AMD Ryzen Threadripper 3970X CPU.

In order to give a meaningful interpretation to resulting numbers, the same speed test has been performed according to the following setups between the two VMs:

1. direct link - no tunnel at all
2. over a GRE tunnel
3. over an OpenVPN tunnel without DCO
4. over an OpenVPN tunnel with DCO

Results

Test	Throughput
Direct	16.1 Gbits/sec
GRE	4.74 Gbits/sec
OpenVPN (no DCO)	713 Mbits/sec
OpenVPN (with DCO)	3.95 Gbits/sec

The first test is used to obtain an upper bound of the setup: this is the maximum possible throughput that can be measured over the link.

The second test, with a GRE tunnel configured, gives us another interesting upper bound as it implements a tunnel with no encryption at all.

The last two lines are the crux of our measurements and show what is the OpenVPN performance without and with DCO. Wihtout DCO means ”using the tun driver”.

As it is possible to see DCO allows OpenVPN to reach a throughput that is more than 5 times faster than the traditional implementation.

Please note that these results were obtained on a modern and fast machine. Plans are to extend this table with tests performed on embedded devices running OpenWrt.

Conclusion

This article introduced OpenVPN Data Channel Offload, a new Linux kernel module aiming at accelerating OpenVPN tunnels. Although other VPN solutions are already present in the Linux kernel, OpenVPN is not going to disappear anytime soon, therefore providing Linux users with a more advanced and modern implementation of the data plane is still a notable goal.

Despite ovpn-dco having reached a certain level of maturity as out-of-tree module, all the points touched in this paper are still open for discussion and (possibly) improvement.