Pushing OpenVPN down the stack: Data Channel Offload

Antonio Quartulli

OpenVPN, Inc.
www.openvpn.net

October 28th, 2022
Netdev 0x16 - Lisbon

About me

Hi! My name is Antonio Quartulli

- **Open Source** enthusiast and developer since ever
- First kernel contributions to **batman-adv** a bit more than 10 years ago: wireless mesh routing protocol
- Later also to **cfg/mac80211**: IBSS (ad-hoc) mode, AP mode
- Started hacking on **OpenVPN** around 6 years ago

OpenVPN what?

- Creates and manages a VPN: peer-to-peer or client/server mode
- Been around for 20 years (originally developed by James Yonan)
- Multi platform support (Linux, Windows, macOS, Android, AIX, FreeBSD, OpenBSD, DragonBSD, ...)
- Allows various authentication methods: certificates, user/password, 2FA, ...
- Supports linking against OpenSSL, LibreSSL, mbedTLS and wolfSSL
- Swiss army knife for network admins

Problematic aspects

- Legacy code
- Fully implemented in userspace
- Relies on the ‘tun’ device driver
- Single threaded

Performance is not on par with current connectivity rates, but tweaking the userspace code has proven to be hard.

Problematic aspects 2

OpenVPN multiplexes two different streams across a connection to a remote endpoint:

- Control Channel: auth, key exchange, param negotiation, ...
- Data Channel: user (encrypted) traffic

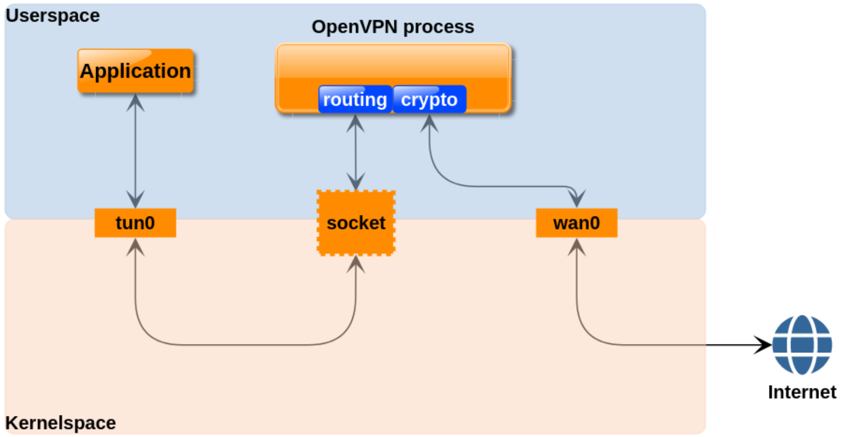
Problematic aspects 2

OpenVPN multiplexes two different streams across a connection to a remote endpoint:

- Control Channel: auth, key exchange, param negotiation, ...
- Data Channel: user (encrypted) traffic

Data Channel traffic has to cross the kernelspace/userspace boundary introducing an important performance penalty

Data Channel flow

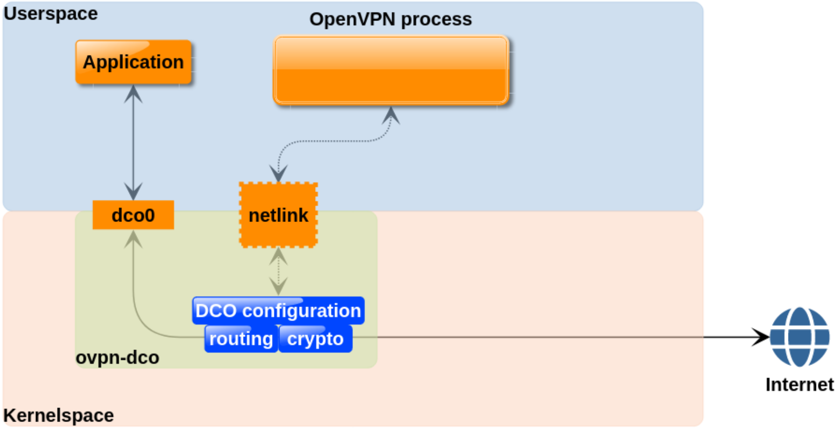


Solution: Data Channel with DCO

How about moving the whole Data Channel handling to kernelspace?
OpenVPN Data Channel Offload (DCO)

Solution: Data Channel with DCO

How about moving the whole Data Channel handling to kernelspace?
OpenVPN Data Channel Offload (DCO)



What is OpenVPN DCO?

- A **virtual device driver** in the Linux kernel that implements the OpenVPN Data Channel
- Encryption/decryption via Crypto API: AES-GCM and ChaCha20Poly1305 supported, but can be easily extended
- Configuration via NetLink (new GENL family: ovpn)
- Interface handling (creation/destruction) via RTNL in v0.1, changed to GENL in v0.2 (WIP)
- Routing uses the system (main) routing table
- Supports TCP and UDP at the transport layer

And the Control Channel?

Still handled in userspace as it was before: **all famous OpenVPN bells and whistles stay out of the kernel**.

After establishing a connection **the socket is passed to DCO**. How are control messages delivered to userspace?

And the Control Channel?

Still handled in userspace as it was before: **all famous OpenVPN bells and whistles stay out of the kernel**.

After establishing a connection **the socket is passed to DCO**. How are control messages delivered to userspace?

v0.1:
Control Channel messages are detected and passed to userspace via NetLink.
Viceversa, Control Channel messages from userspace are also passed to DCO via NetLink and then sent out.

And the Control Channel?

Still handled in userspace as it was before: **all famous OpenVPN bells and whistles stay out of the kernel**.

After establishing a connection **the socket is passed to DCO**. How are control messages delivered to userspace?

v0.1:
Control Channel messages are detected and passed to userspace via NetLink.
Viceversa, Control Channel messages from userspace are also passed to DCO via NetLink and then sent out.

v0.2 (WIP):
After taking ownership of the socket, DCO simply ignores Control Channel messages and let them flow "as usual". **Userspace reads/writes from/to the socket like before**.

NetLink API v0.2 (WIP)

- Interface creation/destruction:
  OVPN_CMD_{NEW,DEL}_IFACE
- Peer management:
  OVPN_CMD_{NEW,SET,GET,DEL}_PEER
- Key management:
  OVPN_CMD_{NEW,DEL}_KEY and OVPN_CMD_SWAP_KEYS

If the process that created the interface disappears, the interface and related state is destroyed.

Userspace maintains its own state of the tunnel (peers, keys status) and uses the NetLink API to synchronyze with DCO.

NetLink events

Peer deletion events (OVPN_CMD_DEL_PEER) are sent from kernelspace to userspace to inform OpenVPN that a peer has disappeared: due to transport error, timeout, user request or shutdown.

We could add more multicast events, but we don't have clear use cases at the moment (i.e. OVPN_CMD_NEW_PEER)

Crypto

- Implemented using the kernel Crypto API
- AEAD: AES-GCM, ChaCha20Poly1305
- Each peer can use a different cipher

Transport

- UDP: implemented using udp_tunnel
- TCP: implemented by changing the socket CBs (similar to kTLS)

RX path

Packets coming from the network are queued in a **ptr-ring**.
The crypto worker is then in charge of picking them one by one and:

1. decrypt
2. decapsulate
3. deliver to the device (using NAPI) or lookup routing table
4. possibly send the packet over the network

TX path

Packets entering the DCO device are queued in a **ptr-ring**.
The crypto worker is then in charge of picking them one by one and:

1. encapsulate
2. encrypt
3. lookup routing table
4. send the packet over the network

Routing

An OpenVPN DCO interface can be configured in 2 modes:

- Peer to Peer (P2P) - aka “dumb tunnel”
- Peer to MultiPeer (P2MP) aka SERVER

Routing

An OpenVPN DCO interface can be configured in 2 modes:

- Peer to Peer (P2P) - aka “dumb tunnel”
- Peer to MultiPeer (P2MP) aka SERVER

In P2MP/Server mode DCO maintains a specific mapping:

- peer VPN IP (v4 or v6) $\ggg$ peer object

Static size: 4k entries.

Routing 2

What if the destination is **behind** the VPN client?

Routing 2

What if the destination is **behind** the VPN client?

```
10.18.0.0/20 via 10.231.203.1 dev dco0
```

Routing 2

What if the destination is **behind** the VPN client?

```
10.18.0.0/20 via 10.231.203.1 dev dco0
```

Destination IP is looked up in the system routing table:

- entry found ⋙ the gateway becomes our destination
- NO entry found ⋙ no change in destination

The destination IP resulting from the above process is looked up in the DCO mapping and a peer is selected

BYOV (Bring Your Own VPN)

OpenVPN DCO **does not care** about the userspace software.

As long as it is properly configured via NetLink it will run and do its job.
(just remember to renew the keys every now and then)

Any userspace application can use DCO to build **its own VPN**.

Links

OpenVPN DCO Repository: https://github.com/OpenVPN/ovpn-dco
It can be used with **OpenVPN2, master branch (soon to be v2.6)**
There is an **OpenWRT feed** for both the above.

Also supported by the **OpenVPN3-Linux client**

Performance

Test: iperf between two Ubuntu 22.04 VMs
Host: AMD Ryzen Threadripper 3970X CPU

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

Next steps

- get merged upstream ("v0.1" is already on netdev ml)
- get faster!
- explore options for HW/NIC offload
- focus testing on embedded devices (i.e. small ARM based routers)
- improve concurrency

Thank you

Thank you for your attention
Questions?