
# Scaling Secure Network Functions

## High-Performance IPsec with FD.io VPP for VNFs and CNFs

Benoît Ganne – bganne@cisco.com
Cisco

FOSDEM 2026 — Saturday, January 31 — Network Track

# About Me

- Technical Leader @ Cisco CloudSec
- VPP Committer @ FD.io
- 20 years pushing packets
  (and still finding new ways to drop them)

**What we run in prod:**

**50K**
tunnels

**7 PB**
per month

**10T**
packets/month

## Why Listen to Me?

Battle scars from scaling IPsec to 50K tunnels – here's what actually works

# What We'll Cover

1. Why IPsec at Scale is Hard
2. FD.io VPP in 60 Seconds
3. What's New in VPP IPsec
4. Policy, P2P, or P2MP?
5. 10,000 Tunnels: The Showdown
6. Breaking the Single-Core Barrier
7. Don't Blow Your Memory Budget
8. Automation Friendly
9. TL;DR

# Why IPsec at Scale is Hard

**VNF/CNF Requirements:**

- SD-WAN: 1000+ tunnels per node
- 5G UPF: N3/N9 interface security
- Fat pipes: 10G+ per tunnel
- High throughput: 40G, 100G+
- Multi-tenancy: VRF per customer

**Traditional Challenges:**

- Kernel IPsec: limited performance
- Kernel modules: incompatible with containers
- Per-tunnel interface overhead
- Memory explosion with VRFs
- No APIs: manual config doesn't scale

**Goal**

Wire-speed IPsec in software, scaling to thousands of tunnels

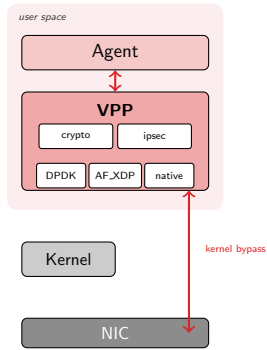
# FD.io VPP in 60 Seconds

**Vector Packet Processing Platform**

- Open-source (Linux Foundation)
- User-space: easy to deploy in VNF/CNF
- API-driven: binary API, REST, gRPC
- Multi-arch: x86, ARM (Graviton, Ampere)

**Why VPP for IPsec?**

- Performance!
- Scale: 10K+ tunnels, async crypto
- Customization: plugin architecture
- Integration: API-driven, cloud-native ready




See other FD.io talks at FOSDEM for deep dives!

# What's New in VPP IPsec (2024–2025)

**Performance Optimizations:**

- Unified crypto+HMAC ops
  - Single operation per packet
- Template-based ESP processing
  - Less overhead
- SA runtime data refactoring
  - Better cache efficiency

**New Features:**

- Full IPv6 NAT-T support
  - UDP encap, RFC 6935/6936
- IPv6 bypass/discard policies
  - Parity with IPv4
- Async handoff queue tuning
  - Tune for your workload

**Active Development**

22 files changed for unified crypto alone – VPP IPsec is actively maintained!

# Policy, P2P, or P2MP?

| | **Policy-Based** | **Route P2P** | **Route P2MP** |
|---|---|---|---|
| **SA selection** | 5-tuple match | interface | next-hop |
| **Interfaces** | none | 1 per tunnel | 1 shared |
| **Routing integration** | no | yes | yes |
| **Dynamic routing** | no | yes | yes |
| **Scale consideration** | flow-cache! | interface count | hash lookup |

**When to use:**

- **Policy-based**: legacy interop, no routing needed, enable flow-cache!
- **Route P2P**: simple ops, moderate tunnel count, dynamic routing
- **Route P2MP**: hub-and-spoke, 100+ spokes, reduced overhead

See backup slides for CLI examples

# 10,000 Tunnels: The Showdown

**Key Findings:**

- Policy without cache: **unusable**
- Flow-cache: **40x faster** (yes, really)
- Both modes scale linearly with cores

**Performance Comparison (Gbps):**

| Cores | Policy | +cache | P2P |
|---|---|---|---|
| 1 | 0.5 | 20 | 24 |
| 2 | 1.3 | 37 | 43 |
| 4 | 2.0 | 49* | 78 |

Test: 10K tunnels, AES-256-GCM, Intel ICX, PDR <0.5%.

Source: FD.io CSIT – `https://csit.fd.io/report/`

* 25ge line rate; P2P on 100ge.

**Takeaway**

**Always enable flow-cache for policy-based IPsec!** Route-based P2P slightly faster but requires more interfaces.

# Breaking the Single-Core Barrier

**Two Problems:**

1. **Fat pipes**: single tunnel $>$ 1 core capacity
2. **Load imbalance**: multiple fat pipes on one core starve other tunnels

**Solution: Async Crypto**

- Decouples packet processing from crypto
- Distributes crypto work across cores
- Prevents noisy neighbor effect
- Enables hardware offload (QAT, etc.)

**Performance (1 tunnel, Gbps):**

| Cores | Sync | Async |
|---|---|---|
| 1 | 28* | – |
| 2 | – | 56 |
| 3 | – | 64 |
| 4 | – | **70** |

Test: 1 tunnel, AES-256-GCM, Intel EMR, PDR $<$0.5%.

Source: FD.io CSIT – `https://csit.fd.io/report/`

* 4 tunnels for sync.

**70 Gbps** on a single tunnel!

# Don't Blow Your Memory Budget

**The Problem:** Each VRF needs its own FIB → memory adds up fast

VPP uses an mtrie for IPv4 FIB lookup. 16-8-8 = 3 levels (65K + 256 + 256 entries). 8-8-8-8 = 4 levels (256 entries each). Smaller root → less memory per VRF.

| 1000 VRFs | Default (16-8-8) | Scale (8-8-8-8) |
|---|---|---|
| Memory | 330 MB | 8 MB |
| Lookup steps | 3 | 4 |

**41x less memory!**

**How:** Compile with `-DVPP_IP_FIB_MTRIE_16=OFF`

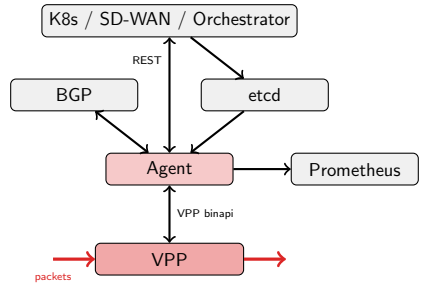
**Trade-off**

One extra lookup step for 41x memory savings – worth it at scale

# Automation Friendly

**API Options:**

- **Binary API** – Piloting VPP
- **GoVPP** – Go bindings
  - `go.fd.io/govpp`
  - GoBGP for routing – see later talk!
- **Python** – scripting & test
- **HTTP/REST** – JSON over HTTP
- **etcd** – K8s-native config store




## Cloud-Native Ready

No kernel modules – runs in containers. API-driven – fits GitOps workflows.

# TL;DR

1. **Choose the right IPsec mode**
   - Policy+flow-cache: no interface overhead, good for many tunnels
   - Route-based P2P/P2MP: dynamic routing, scale
2. **Use async crypto scheduler**
   - Prevents noisy neighbor, enables HW offload
   - 70 Gbps single tunnel (2.5x sync)
3. **Plan for memory at scale**
   - 8-8-8-8 mtrie: 41x reduction for 1000+ VRFs
4. **Integrate via API**
   - GoVPP for K8s/orchestration, Python for scripting
   - No kernel modules – container-friendly

**Resources:**

- FD.io: `https://fd.io`
- CSIT: `https://csit.fd.io` — GoVPP: `https://go.fd.io/govpp`

# Questions?

Thank you!


Benoît Ganne – bganne@cisco.com

https://fd.io

# Backup: Policy-Based IPsec CLI

```
# Create SAs
ipsec sa add 10 spi 1000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef tunnel src 1.1.1.1 dst 2.2.2.2
ipsec sa add 20 spi 2000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef inbound tunnel src 2.2.2.2 dst 1.1.1.1

# Create SPD and bind to interface
ipsec spd add 1
set interface ipsec spd local0 1

# Add policies
ipsec policy add spd 1 outbound priority 100 action protect
    local-ip-range 10.0.0.0-10.0.0.255 remote-ip-range 10.1.0.0-10.1.0.255 sa 10
ipsec policy add spd 1 inbound priority 100 action protect
    local-ip-range 10.0.0.0-10.0.0.255 remote-ip-range 10.1.0.0-10.1.0.255 sa 20
```

# Backup: Route-Based IPsec CLI

```
# Create SAs
ipsec sa add 10 spi 1000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef tunnel src 1.1.1.1 dst 2.2.2.2
ipsec sa add 20 spi 2000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef tunnel src 2.2.2.2 dst 1.1.1.1 inbound

# Create IPsec interface
ipsec itf create instance 0

# Protect interface with SAs
ipsec tunnel protect ipsec0 sa-in 20 sa-out 10

# Route traffic via interface
ip route add 10.1.0.0/24 via ipsec0
```

# Backup: P2MP IPsec CLI

```
# Create SAs for multiple peers
ipsec sa add 10 spi 1000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcde0 tunnel src 1.1.1.1 dst 2.2.2.2
ipsec sa add 11 spi 2000 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcde0 tunnel src 2.2.2.2 dst 1.1.1.1 inbound
ipsec sa add 20 spi 1001 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcde1 tunnel src 1.1.1.1 dst 3.3.3.3
ipsec sa add 21 spi 2001 crypto-alg aes-gcm-256
    crypto-key 0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcde1 tunnel src 3.3.3.3 dst 1.1.1.1 inbound

# Create P2MP IPsec interface
ipsec itf create instance 0 p2mp

# Protect with multiple peers
ipsec tunnel protect ipsec0 sa-in 11 sa-out 10 2.2.2.2
ipsec tunnel protect ipsec0 sa-in 21 sa-out 20 3.3.3.3

# Route to peers via interface
ip route add 10.1.0.0/24 via 2.2.2.2 ipsec0
ip route add 10.2.0.0/24 via 3.3.3.3 ipsec0
```