Multi-core IPsec tunnels

Daniel Xu, Vlad Dumitrescu, Antony Antony

Additional credits

- Kernel work ~3-4 years ago @Netdev 0x13 in Prague.
- Many thanks to Steffen Klassert for xfrm patches
- Tobias Brunner : strongSwan support
- Paul Wouters : IETF standardization, and testing using libreswan
- Sowmini Varadhan : initial use case
- Benedict Wong and Tuomo Soini : hacking and testing
- Jonathan Lemon : ENA driver XDP multi-buffer support

Background

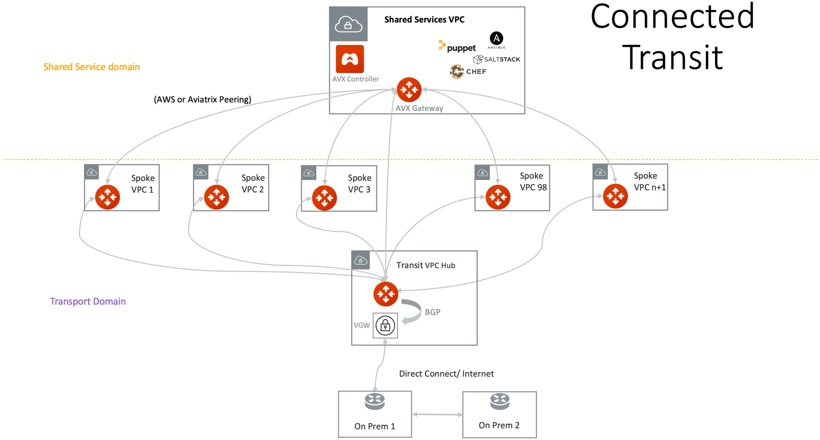
- IPsec tunnels have well known scalability limitations
  - Crypto state, counters, and sequence numbers cannot be efficiently shared across cores
- Link speeds vastly outpacing single tunnel performance improvements
- Would like to take advantage of modern multi-core systems

Typical topology

```
+------------------+           +------------------+
| Private Subnet 1 |           | Private Subnet 2 |
+--------+---------+           +--------+---------+
         |                              |
         |                              |
         |                              |
+--------v--------+             +-------v---------+
|   Gateway 1     |             |   Gateway 2     |
+--------+--------+             +-------+---------+
         |                              |
         |                              |
         |                              |
         +--------------+  +------------+
                        |  |
                    IPSec Tunnel
                  +---Internet---+
```

Aviatrix topology

Connected Transit

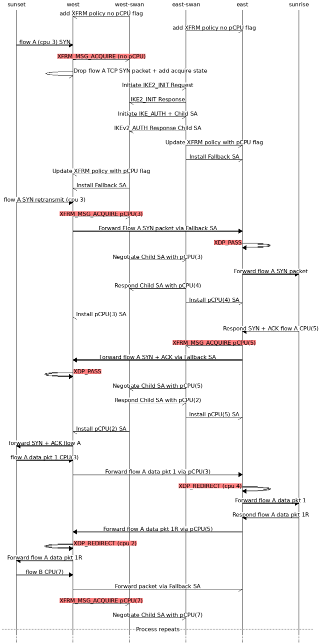


pCPU IPsec tunnel design

- Negotiate a pair of SAs for each CPU
  - On demand and sender driven
- On TX, the pCPU SA is chosen based on current CPU
- On RX, expect a given pCPU SPI to always land on same CPU
  - Hardware RSS or software RSS (XDP_REDIRECT)
- If RX and TX constraints are met: lockless operation and linear scaling

End to end sequence

dxuuu.xyz/r/ipsec-pcpu.png



sunset
west
west-swan
east-swan
east
sunrise

add XFRM policy no pCPU flag

add XFRM policy no pCPU flag

flow A (cpu 3) SYN

XFRM_MSG_ACQUIRE (no pCPU)

Drop flow A TCP SYN packet + add acquire state

Initiate IKE2_INIT Request

IKE2_INIT Response

Initiate IKE_AUTH + Child SA

IKEv2_AUTH Response Child SA

Update XFRM policy with pCPU flag

Install Fallback SA

Update XFRM policy with pCPU flag

Install Fallback SA

flow A SYN retransmit (cpu 3)

XFRM_MSG_ACQUIRE pCPU(3)

Forward Flow A SYN packet via Fallback SA

XDP_PASS

Negotiate Child SA with pCPU(3)

Forward flow A SYN packet

Respond Child SA with pCPU(4)

Install pCPU(4) SA

Install pCPU(3) SA

Respond SYN + ACK flow A CPU(5)

XFRM_MSG_ACQUIRE pCPU(5)

Forward flow A SYN + ACK via Fallback SA

XDP_PASS

Negotiate Child SA with pCPU(5)

Respond Child SA with pCPU(2)

Install pCPU(5) SA

Install pCPU(2) SA

forward SYN + ACK flow A

flow A data pkt 1 CPU(3)

Forward flow A data pkt 1 via pCPU(3)

XDP_REDIRECT (cpu 4)

Forward flow A data pkt 1

Respond flow A data pkt 1R

Forward flow A data pkt 1R via pCPU(5)

XDP_REDIRECT (cpu 2)

Forward flow A data pkt 1R

flow B CPU(7)

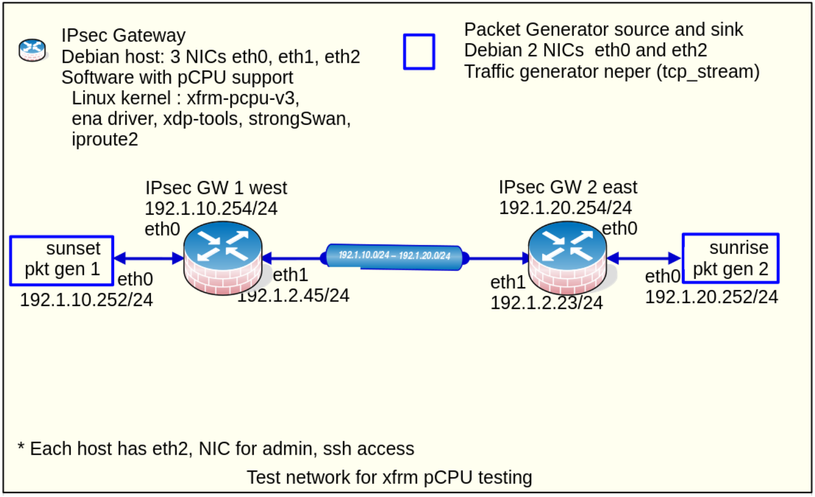
Forward packet via Fallback SA

XFRM_MSG_ACQUIRE pCPU(7)

Negotiate Child SA with pCPU(7)

Process repeats

Experimental setup



IPsec Gateway
Debian host: 3 NICs eth0, eth1, eth2
Software with pCPU support
Linux kernel : xfrm-pcpu-v3,
ena driver, xdp-tools, strongSwan,
iproute2

Packet Generator source and sink
Debian 2 NICs eth0 and eth2
Traffic generator neper (tcp_stream)

IPsec GW 1 west
192.1.10.254/24
eth0

IPsec GW 2 east
192.1.20.254/24
eth0

sunset
pkt gen 1
eth0
192.1.10.252/24

eth1
192.1.2.45/24

192.1.10.0/24 – 192.1.20.0/24

eth1
192.1.2.23/24

sunrise
pkt gen 2
eth0
192.1.20.252/24

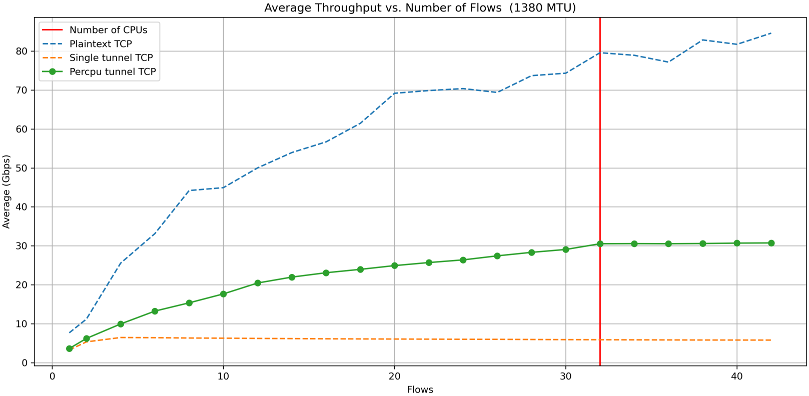
* Each host has eth2, NIC for admin, ssh access

Test network for xfrm pCPU testing

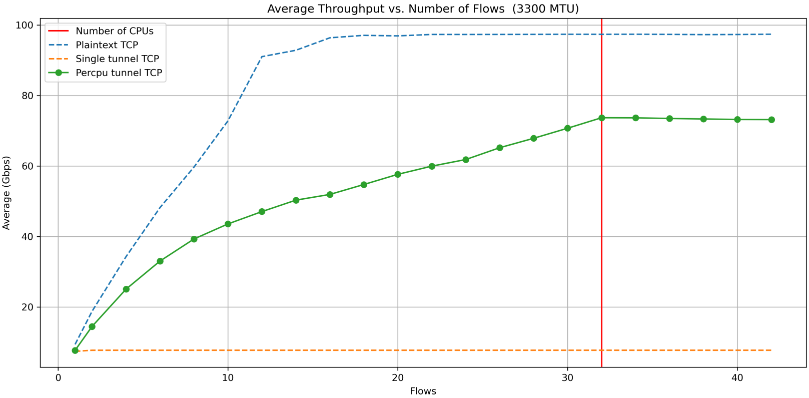
Experimental setup details

- All hosts are EC2 `c6in.16xlarge`
    - 32 physical cores (hyperthreading disabled)
    - 100 Gbps instance bandwidth
    - 10 Gbps single flow limit
    - 16 combined rx/tx queues
- xfrm pcpu patches applied to 6.5.6 Debian sid kernel
- ENA patches applied to prevent XDP queue halving and for jumbo frames
- GRO disabled on all dataplane interfaces (more on this later)
- XDP_REDIRECT used for steering
    - SPI for rx
    - sport/dport for plaintext tx
- `neper` (`tcp_stream`) used for traffic
- UDP encap used to overcome single flow limit
    - AWS does not differentiate ESP flows based on SPI :(

Results (1/3)

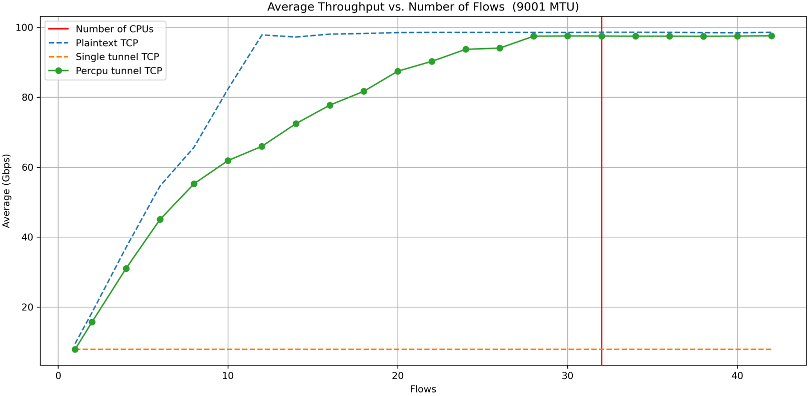

Average Throughput vs. Number of Flows (1380 MTU)
Number of CPUs
Plaintext TCP
Single tunnel TCP
Percpu tunnel TCP
Average (Gbps)
0
10
20
30
40
50
60
70
80
Flows
0
10
20
30
40

Results (2/3)



Average Throughput vs. Number of Flows (3300 MTU)
Number of CPUs
Plaintext TCP
Single tunnel TCP
Percpu tunnel TCP
Average (Gbps)
100
80
60
40
20
0
10
20
30
40
Flows

Results (3/3)



Average Throughput vs. Number of Flows (9001 MTU)
Number of CPUs
Plaintext TCP
Single tunnel TCP
Percpu tunnel TCP
Average (Gbps)
100
80
60
40
20
0
10
20
30
40
Flows

Near-term improvements

- XDP cpumap GRO support
  - To help batch up plaintext flows to hand to xfrm
  - Big expected win here
  - Patches already exist!
- xfrm pcpu tx contention
  - Unexpected appearanes in cpu profile:
    - `xfrm_resolve_and_create_bundle()`
    - `xfrm_state_find()`

It takes a village

Changes in:

- kernel xfrm
- kernel bpf
- Amazon ENA driver
- strongSwan
- xdp-tools
- iproute2
- IETF