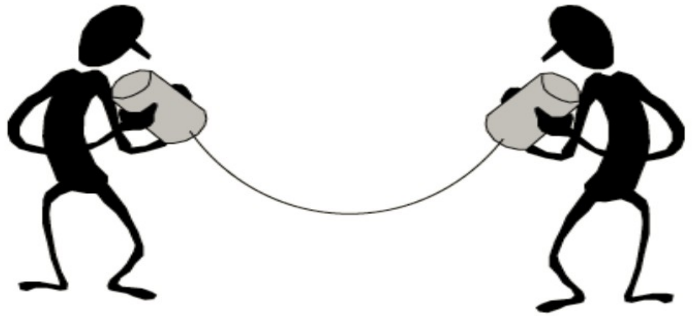
Notes on IPsec Offload using Intel 10Gbe

Netdev 2.2

- Shannon Nelson
- Oracle Corp

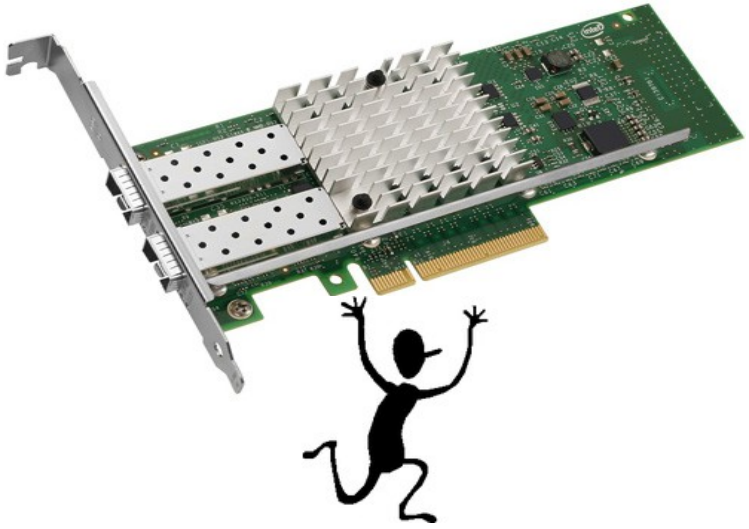
- November 2017

(updated post-conference)



TL;DR

- Niantic and family have IPsec HW offload
  - Intel 10GbE NIC and LOM – 82599, x520, x540, x550
- Driver patches are in development
- Basic Encryption/Decryption offload is working

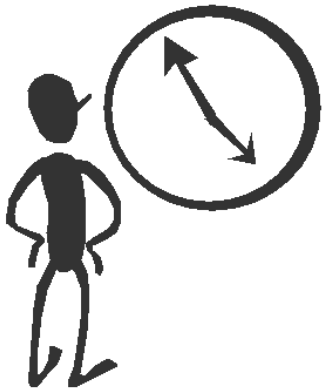


Why Do I Care?

- Oracle customers already have the NICs
- No new $$'s needed to help our customers make use of it
- Something to experiment on while driving further development
- I used to work for Intel on the ixgbe and related drivers
- Sowmini told me to

NIC Timeline

- 2009 – initial NIC and driver release
  - no Linux IPsec offload available in the network stack
  - some support in Windows PROset drivers
- 2016 – HW offload work in XFRM
  - early ixgbe hw offload work from Joshua Hay
- 2017 – initial HW offload merged into upstream kernel
  - support in Mellanox Innova mlx5e released
  - ixgbe support in progress



Background work

- Studied the public data sheet
- Reviewed Mellanox and XFRM code for hints
- Studied early attempt at adding to ixgbe
- Cajoled Intel for guidance

Theory – Engine Setup

- Set NETIF_F_HW_ESP in netdev features at probe
- Don't waste chip power when not offloading
  - Don't start the engine until first SA is offloaded
  - Stop the engine when last SA is removed
- Bits and pieces
  - SW tables to track HW table contents for reload on device resets
  - Set netdev->xfrmdev_ops for offload functions
  - Stop Rx and Tx datapaths and wait to drain
  - Increase IFG and tweak buffer-full threshold
  - Enable offload engine then enable SA lookups

Theory – SA Storage

- NDO ops: xdo_dev_state_add, xdo_dev_state_delete, ...
- Chip tables
  - Tx: 128 bit Key, 32 bit Salt (1024)
  - Rx: Key, Salt, SPI, Mode, IPidx (1024) ; IP addrs (128)
    - CAM (content-addressable mem) for faster IPaddr/SPI lookups – not cleared on reset
- 2-step Load/Read Operation
  - Load a bounce register
  - Write index register to move input to table slot
  - Reverse for reading

Theory – Tx Offload

- XFRM Stack sets up packet data
  - Inserts IPsec header & trailer into packet, no encryption
  - Sets up skb->sp with SA selection handle
- Driver writes Tx Context Descriptor
  - Bits for ESP/AH, Encryption, trailer length, Tx SA Key table index
- Driver writes Tx Send Descriptor
  - Bit for IPsec offload triggers use of IPsec context info
  - Engine encrypts data, twiddles ICV, csum, counter fields as needed

Theory – Rx Offload

- Engine watches for IPsec header in packets
  - Search for for SA Key using SPI, dest IP, and IPv4/6
  - Decrypt, check resulting ICV
  - Set status bits to signal IPsec found and any decode errors
- Driver reads Rx Writeback descriptor status
  - IPsec found, ESP/AH, and 2 bits for success/error status
  - No indication of SA used, so SW also must do lookup
  - Add offload status to skb->sp
- XFRM Receive strips off ESP header & trailer

Current Status – It's Alive

- IPv4 only so far
- aead only with 128bit rfc4106(gcm(aes))
- 1024 keys with 256 IP addresses
- Checksum and TSO offload support not yet implemented
- Performance not worth measuring yet
  - (post conference note: 7+ Gbps on a 10GbE connection with simple iperf, no csum, no tso)

Challenges – XFRM API

- Error handling details
  - Failed SA add didn't delete (unwind) offload from driver
    - c5d4d7d8316 xfrm: Fix deletion of offloaded SAs on failure.
  - Failed SA offload messed up the reference count
    - 67a63387b14 xfrm: Fix negative device refcount on offload failure.

Challenges – Documentation

- Minimal-to-none XFRM API documentation
  - https://netdevconf.org/1.2/slides/oct7/08_2_IPsec_workshop_Boris_Pismenny.pdf
- IPsec in iproute2
  - Difficult and incomplete man pages and command line help
- 82599, x540 datasheets
  - Typical datasheet – sketchy info, takes experimentation and interpretation
  - Gets better with x540 and x550

Challenges – Documentation

- Example – byte order of key, spi, and ip address
  - What order do I use in the ip command?
  - What order do they come into the driver from XFRM api?
  - What order do they need to be installed in the chip tables?
  - How will these change between SPARC and Intel arch?

Challenges – legacy driver

- Older 82598 definitions clash with newer 82599 bits
  - Rx error status descriptor fields
  - Tx descriptor bits
  - IPsec register fields

Challenges – NIC weirdness

- Registers not cleared on reset
  - CAM – content addressable memory for faster lookups
- Experimentation needed for order of operations
  - Getting the offload setup steps in just the right
- Support from Intel has been slow
  - Many many thanks to Jesse Brandeburg for stepping in to help

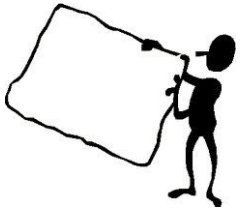
Performance

- tbd
  - (post conference note:
    10Gbe connection with simple iperf
    SW ipsec: ~300 Mbps
    HW offload: 7+ Gbps, no csum, no tso)

To Do

- Enable checksum and TSO support
- Add IPv6 support
- Add tunnel support
- Replace simple table lookup with hashed lists
- Fix up documentation
- Performance comparisons with QAT

Test setup example – tcp.all



- Left:
  - ip x p add dir out src 14.0.0.52/24 dst 14.0.0.70/24 proto tcp tmpl proto esp src 14.0.0.52 dst 14.0.0.70 spi 0x07 mode transport reqid 0x07
  - ip x p add dir in src 14.0.0.70/24 dst 14.0.0.52/24 proto tcp tmpl proto esp dst 14.0.0.52 src 14.0.0.70 spi 0x07 mode transport reqid 0x07
  - ip x s add proto esp src 14.0.0.52 dst 14.0.0.70 spi 0x07 mode transport reqid 0x07 replay-window 32 aead 'rfc4106(gcm(aes))' 0x44434241343332312423222114131211f4f3f2f1 128 sel src 14.0.0.52/24 dst 14.0.0.70/24 proto tcp
  - ip x s add proto esp dst 14.0.0.52 src 14.0.0.70 spi 0x07 mode transport reqid 0x07 replay-window 32 aead 'rfc4106(gcm(aes))' 0x44434241343332312423222114131211f4f3f2f1 128 sel src 14.0.0.70/24 dst 14.0.0.52/24 proto tcp
- Right:
  - ip x p add dir out src 14.0.0.70/24 dst 14.0.0.52/24 proto tcp tmpl proto esp src 14.0.0.70 dst 14.0.0.52 spi 0x07 mode transport reqid 0x07
  - ip x p add dir in src 14.0.0.52/24 dst 14.0.0.70/24 proto tcp tmpl proto esp dst 14.0.0.70 src 14.0.0.52 spi 0x07 mode transport reqid 0x07
  - ip x s add proto esp src 14.0.0.70 dst 14.0.0.52 spi 0x07 mode transport reqid 0x07 replay-window 32 aead 'rfc4106(gcm(aes))' 0x44434241343332312423222114131211f4f3f2f1 128 sel src 14.0.0.70/24 dst 14.0.0.52/24 proto tcp **offload dev eth4 dir out**
  - ip x s add proto esp dst 14.0.0.70 src 14.0.0.52 spi 0x07 mode transport reqid 0x07 replay-window 32 aead 'rfc4106(gcm(aes))' 0x44434241343332312423222114131211f4f3f2f1 128 sel src 14.0.0.52/24 dst 14.0.0.70/24 proto tcp **offload dev eth4 dir in**

Other Tidbits

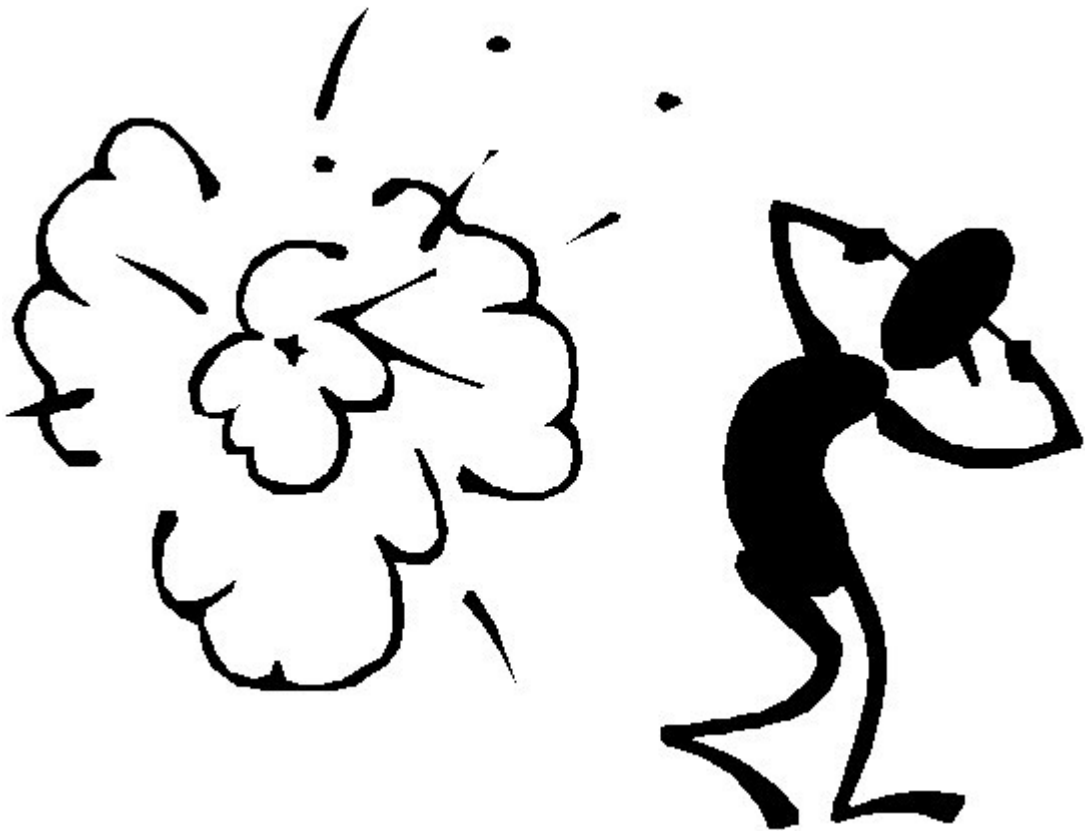
https://libreswan.org/wiki/Benchmarking_and_Performance_testing

- Some benchmarking info on Niantic + IPsec with no offload; also references using QAT for some encrypt/decrypt offload help but with different encryption than Niantic IPsec; also has to send data across the PCI bus multiple times

82599 & x540 used in DPDK/Lua implementation for ipsec offload

- suggests 10Gbe line speed possible with engine https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/CloudNet2016.pdf

Questions?



http://weclipart.com/screen+bean+people+clipart