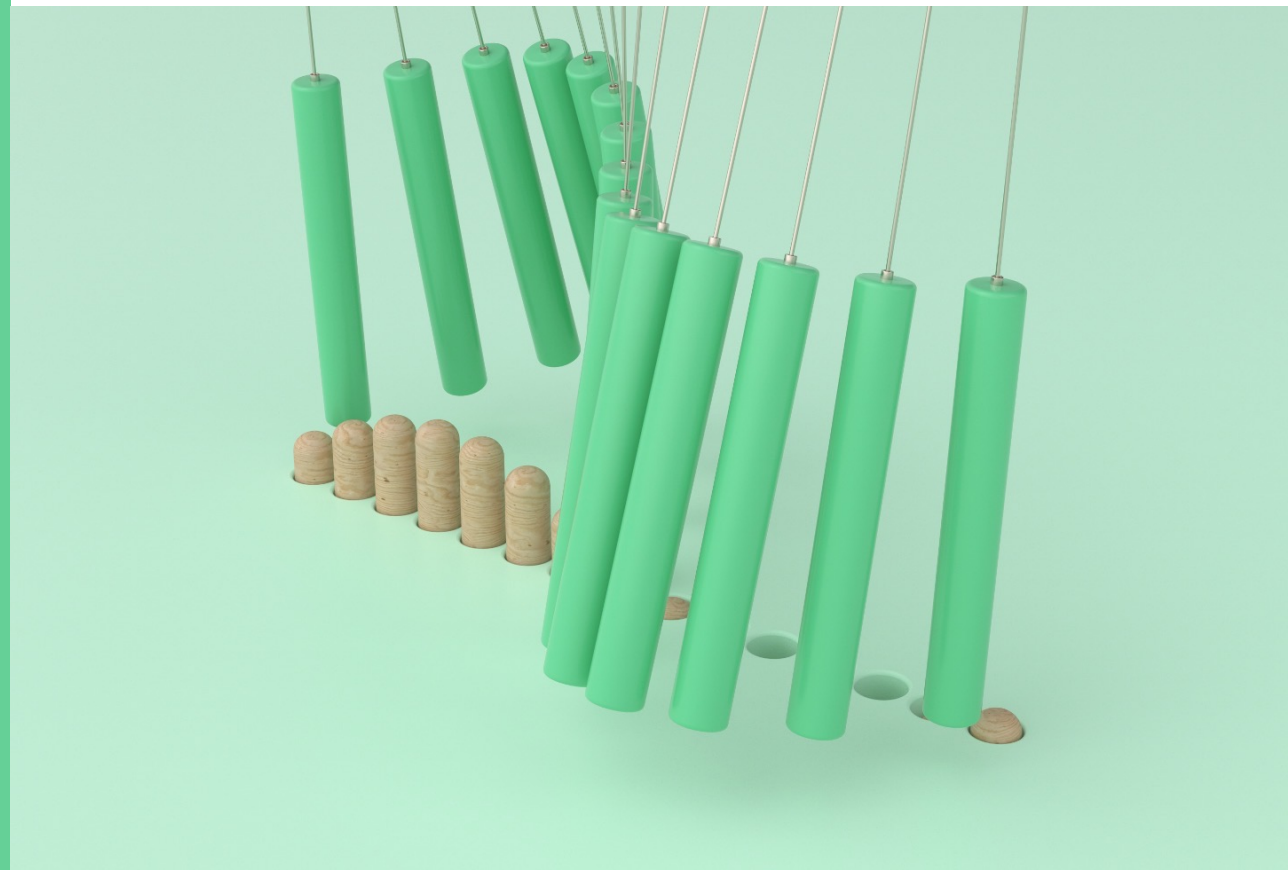
IFLIB @ ONTAP

(Performance Journey for Intel 10G - swiWarp)

Sai Rajesh Tallamraju
stallamr@netapp.com

ONTAP NIC Drivers, NetApp
June 09,2021



NetApp at a glance

Industry-leading cloud data services

Cloud storage

Compute operations

Cloud controls

Cloud services and analytics

Industry-leading storage systems and software

Flash and hybrid storage

Object storage

Converged and hybrid cloud infrastructure

Protection and security

Enterprise solutions

Industry-leading solutions with an open ecosystem of partners

- **Global** cloud-led, data-centric software company
- **Founded in 1992**, headquartered in Sunnyvale, California
- **Fortune 500 company** (NASDAQ: NTAP)
- **$5.41B FY20 revenue**
- **38,000+ customers** around the world
- **Industry-leading cloud** and **enterprise data center** solutions and services
- **10,800+ employees + 5,400 partners** helping customers thrive in a hybrid multicloud world
- **98 offices** in **over 30 countries**

ONTAP is FreeBSD

https://youtu.be/Ei6brWgr5N8

- 1990's—ONTAP was "embedded"
  - Custom hardware
  - Monolithic kernel
  - No user-space

- 2000's—ONTAP needed to "scale out"
  - Off-the-shelf hardware
  - Virtual platforms
  - Clusterize all the things

- Answer: FreeBSD
  - Open licensing model.
  - Cluster management in user-space.
  - ONTAP magic in "just" a couple kernel modules
  - 1000s of customizations to FreeBSD itself

Slide credits: Alexander Sideropoulos, November 13, 2020.

IFLIB Journey

- Introduction of IFLIB
  - IFLIB is a framework for network drivers in FreeBSD.
  - Move large amount of boilerplate code that is often duplicated into a single framework.
  - Network drivers focus on hardware specific code.
  - Intel 1G/10G/40G drivers adopted IFLIB.
  - Ontap journey with IFLIB started @ SVN r327031

- But…
  - A complete driver re-write.
  - Less documentation (manpage and wiki)
  - IFLIB + NetApp customization – significant effort.
  - Steep learning curve.
  - Lacks feature-parity initially.
  - Complex code – path length increase.
  - After decent stability – uphill battle with performance now!
  - Standard performance measurements fare better than non-IFLIB version.
  - But sw-iWarp sees huge spike in latency (mbuf).

Experiments / changes

- Initial theory – LRO & TSO sizes are way below in comparison to our baseline.

- Attempt to match or exceed LRO & TSO sizes.
  - Static interrupt rate
    - Default - IXGBE_LOW_LATENCY - setenv hw.ix.max_interrupt_rate 31250
    - IXGBE_AVE_LATENCY - setenv hw.ix.max_interrupt_rate 10000
    - IXGBE_BULK_LATENCY - setenv hw.ix.max_interrupt_rate 3333
    - No improvement in latency.

  - Change LRO size (16K vs 13K vs 17K)
    - https://reviews.freebsd.org/D27465
    - Stop aggressive checking for available Rx descriptors in iflib_rxeof().
    - Rely on IRQ-driven Rx descriptors.
    - Minimal latency improvement.

  - Change TSO Segment size (2K vs 4K vs 16K)
    - Alter isc_tx_tso_segsize_max and isc_tso_maxsegsize
    - No latency improvement.

Experiments / changes - continued

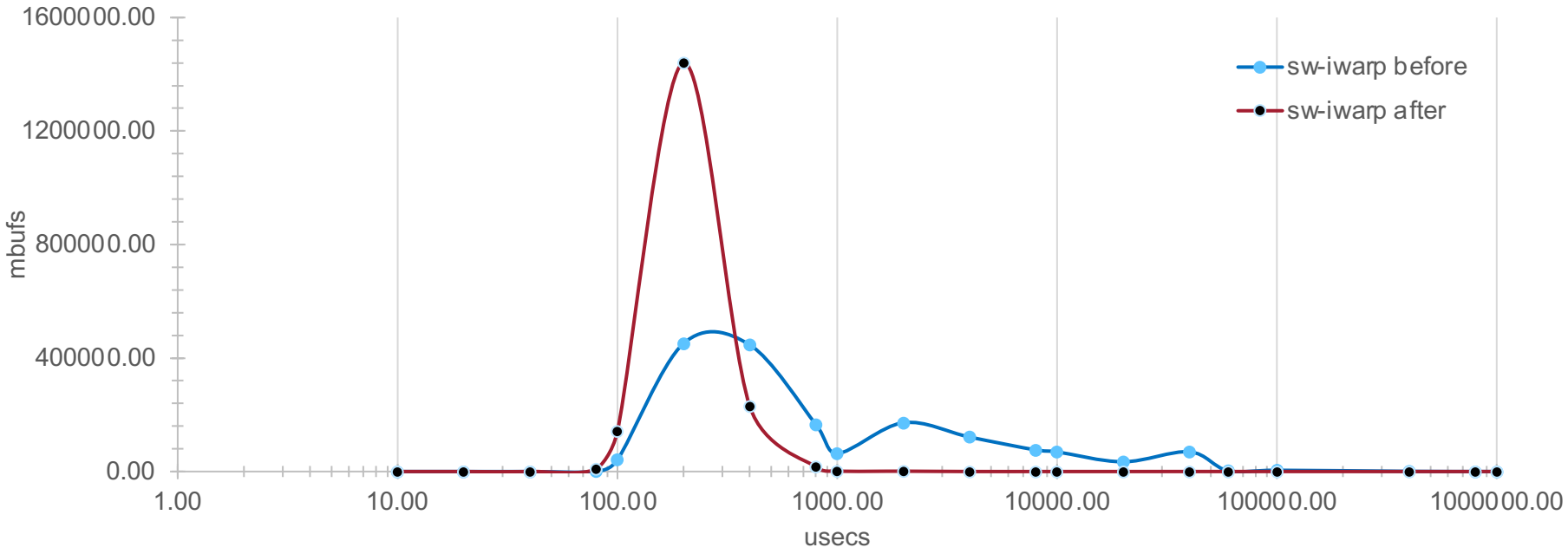
- LRO & TSO sizes matched to baseline. AIM is back but still no improvement.

- Adaptive Interrupt Moderation
  - Bring back AIM from BSD11.
  - Start with only Rx accounting and expand to include Tx accounting too.
  - No significant improvement.
  - https://reviews.freebsd.org/D27344
  - https://reviews.freebsd.org/D30094
  - https://reviews.freebsd.org/D30155

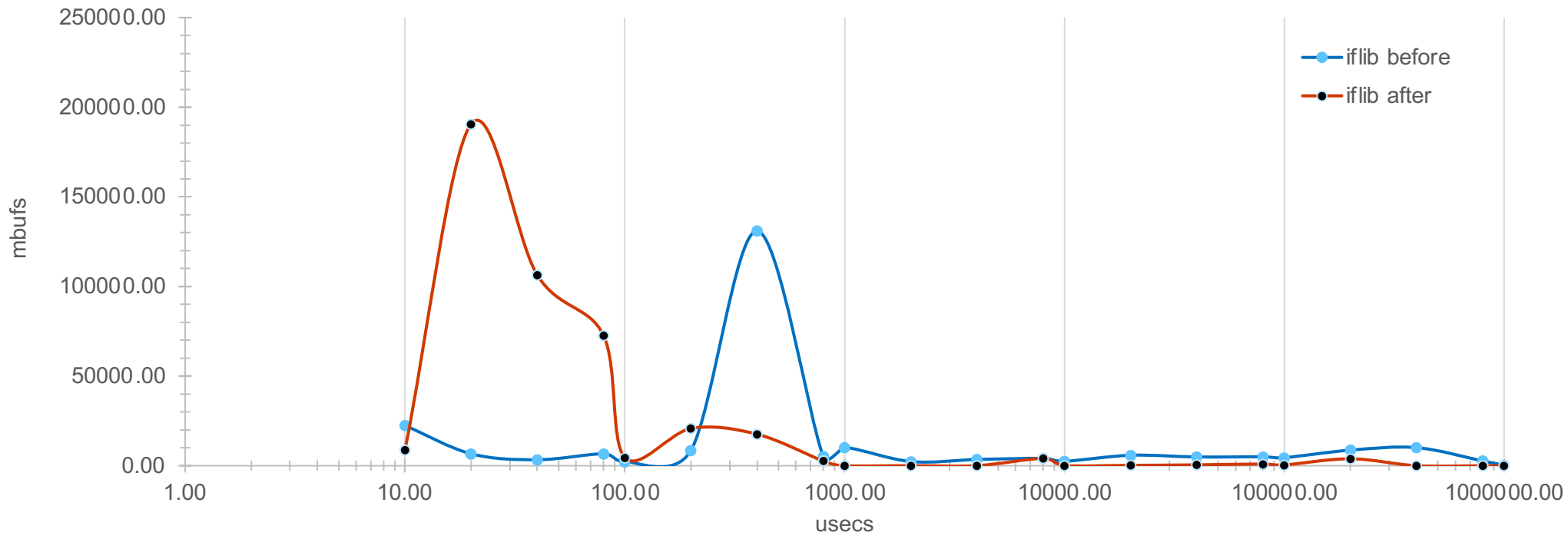
- Unique IRQ for Rx and Tx
  - Create unique IRQ vector for each Rx and Tx queues.
  - No significant improvement.

- Got down to basics – mbuf.
  - Track down per mbuf/mbuf-chain latency at different stages in the networking stack.
  - Boom!

sw-iWarp – Mbuf – Life Cycle



IFLIB – Mbuf – Life Cycle



Breakthrough

- The key is to separate Tx reclaim processing from Tx processing.

- Following are the changes made..
  - Reclaim Tx descriptors in iflib_fast_intr_rxtx().
  - Make iflib_txq_drain() perform only encap() i.e, enqueue to HW queues.
  - Remove iflib_completed_tx_reclaim() from iflib_txq_drain().
    - Ideally, we want to reclaim completed Tx descriptors on Tx path too.
    - But IFLIB Tx processing relies on lockless MP-Ring.
    - Tx processing outside MP-Ring context results in no-traffic / queue-hung situation.
  - Disable txq_max_db_deferred() and txq_max_rs_deferred().
  - Rely on HW AIM (Adaptive Interrupt Moderation) instead.

Thanks to Frank Yang for overseeing all the effort.
Thanks to Mark Johnston for committing few of these changes back to community.

Learnings

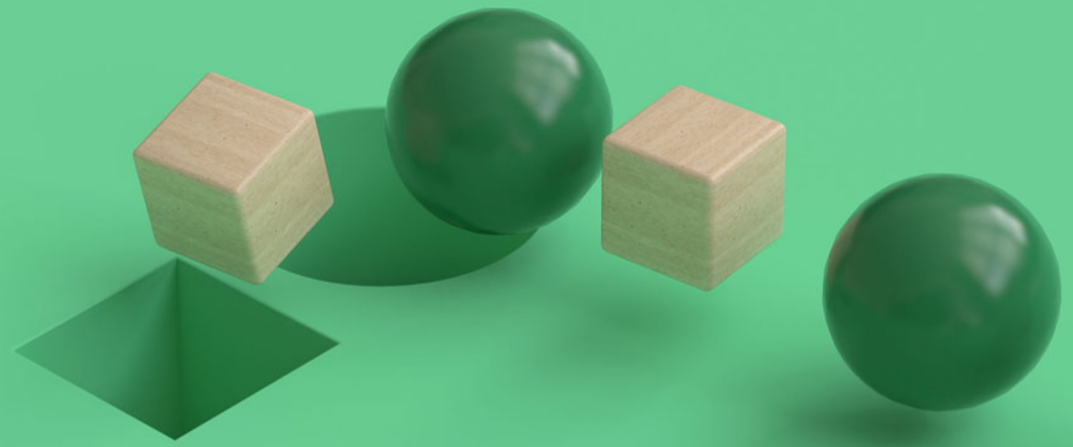
- For Intel 1G/10G/40G drivers – Rx and Tx queues are still tied together.
- IFLIB created IRQ only for the Rx side.
- Tx side is a soft-irq – driven by task enqueue.
- Inherently, Tx packet completion too generate IRQ.
- With AIM, we need to moderate based on both Rx and Tx accounting. Else too many interrupts.
- IFLIB gave up on HW Interrupt Moderation and relied on its own moderation.
  - DB (doorbell) defer – Update HW PIDX tail only after X entries.
  - RS (result status) defer – Notify only after X entries.
  - Inherently application waits bit-longer to have data sent and data completion notification.

- IFLIB relies on MP Ring – a lockless buffer ring.
  - Cannot have reclaim Tx descriptor in Rx and Tx paths.

Thanks to Frank Yang and John Hainline for overseeing all the effort.
Thanks to Mark Johnston for committing few of these changes back to community.

Send ideas and questions to:
stallamr@netapp.com

Backup



sw-iWarp - Mbuf latency

Before

siw_histogram_10usecs: 0
siw_histogram_20usecs: 0
siw_histogram_40usecs: 0
siw_histogram_80usecs: 648
siw_histogram_100usecs: 41217
siw_histogram_200usecs: 451303
siw_histogram_400usecs: 446488
siw_histogram_800usecs: 165821
siw_histogram_1msecs: 63744
siw_histogram_2msecs: 172118
siw_histogram_4msecs: 121491
siw_histogram_6msecs: 76001
siw_histogram_8msecs: 69207
siw_histogram_10msecs: 34222
siw_histogram_40msecs: 69579
siw_histogram_60msecs: 2306
siw_histogram_100msecs: 4462
siw_histogram_400msecs: 634
siw_histogram_800msecs: 2
siw_histogram_1secs: 0
siw_histogram_2secs: 0
siw_histogram_3secs: 0
siw_histogram_4secs: 0

After

siw_histogram_10usecs: 0
siw_histogram_20usecs: 0
siw_histogram_40usecs: 0
siw_histogram_80usecs: 8810
siw_histogram_100usecs: 141202
siw_histogram_200usecs: 1440164
siw_histogram_400usecs: 230140
siw_histogram_800usecs: 16930
siw_histogram_1msecs: 1800
siw_histogram_2msecs: 1174
siw_histogram_4msecs: 42
siw_histogram_6msecs: 21
siw_histogram_8msecs: 13
siw_histogram_10msecs: 12
siw_histogram_40msecs: 70
siw_histogram_60msecs: 17
siw_histogram_100msecs: 63
siw_histogram_400msecs: 11
siw_histogram_800msecs: 0
siw_histogram_1secs: 0
siw_histogram_2secs: 0
siw_histogram_3secs: 0
siw_histogram_4secs: 0

Credits: Wenjun Wang, Sudharshan Varadarajan

IFLIB – Mbuf latency

Before

iflib_histogram_10usecs: 22441
iflib_histogram_20usecs: 6662
iflib_histogram_40usecs: 3325
iflib_histogram_80usecs: 6632
iflib_histogram_100usecs: 2348
iflib_histogram_200usecs: 8572
iflib_histogram_400usecs: 131012
iflib_histogram_800usecs: 5155
iflib_histogram_1msecs: 10206
iflib_histogram_2msecs: 2321
iflib_histogram_4msecs: 3493
iflib_histogram_8msecs: 4076
iflib_histogram_10msecs: 2482
iflib_histogram_20msecs: 5835
iflib_histogram_40msecs: 4888
iflib_histogram_80msecs: 4980
iflib_histogram_100msecs: 4498
iflib_histogram_200msecs: 8703
iflib_histogram_400msecs: 10087
iflib_histogram_800msecs: 2652
iflib_histogram_1secs: 38
iflib_histogram_2secs: 173
iflib_histogram_4secs: 53
iflib_histogram_8secs: 1
iflib_histogram_10secs: 0
iflib_histogram_20secs: 0
iflib_histogram_40secs: 1

After

iflib_histogram_10usecs: 8676
iflib_histogram_20usecs: 190600
iflib_histogram_40usecs: 106326
iflib_histogram_80usecs: 72761
iflib_histogram_100usecs: 4556
iflib_histogram_200usecs: 20764
iflib_histogram_400usecs: 17576
iflib_histogram_800usecs: 2655
iflib_histogram_1msecs: 32
iflib_histogram_2msecs: 39
iflib_histogram_4msecs: 55
iflib_histogram_8msecs: 4076
iflib_histogram_10msecs: 104
iflib_histogram_20msecs: 225
iflib_histogram_40msecs: 486
iflib_histogram_80msecs: 934
iflib_histogram_100msecs: 437
iflib_histogram_200msecs: 3873
iflib_histogram_400msecs: 3
iflib_histogram_800msecs: 0
iflib_histogram_1secs: 0
iflib_histogram_2secs: 0
iflib_histogram_4secs: 0
iflib_histogram_8secs: 0
iflib_histogram_10secs: 0
iflib_histogram_20secs: 0
iflib_histogram_40secs: 0

Credits: Wenjun Wang, Sudharshan Varadarajan

