MIT HAYSTACK OBSERVATORY

Pushing the Mark6 to 60Gbps and Beyond with DPDK

J. Barrett

MIT Haystack Observatory
IVTW Meeting IX

October 23, 2024

Outline

MIT HAYSTACK OBSERVATORY

1. Mark6 background
2. Updated dplane architecture and DPDK (Data Plane Development Kit)
3. Testing and results
4. Future work/outlook

Mark6 Background

MIT HAYSTACK OBSERVATORY

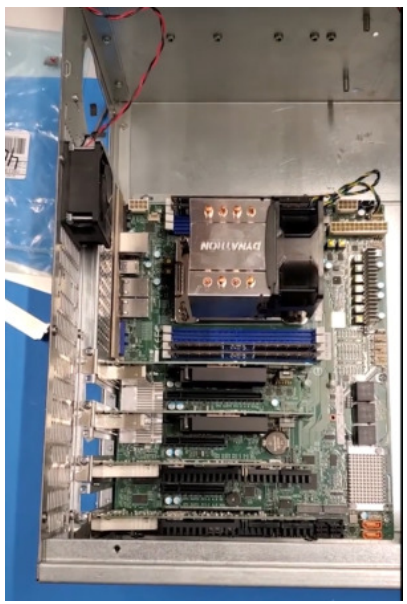
1. Development initiated in the late '00s
2. Designed to support VLBI recording to readily swapped disk-modules
3. Nominally supports continuous recording at 4Gbps per disk-module (up to 16Gbps)
4. Hardware is supplied commercially by the Conduant Corp.
5. Software is split into two components:
   1. cplane - the control plane handles user interaction via VSI-S command set, and system configuration
   2. dplane - the data plane handles network-to-disk data movement

- Commodity computer hardware in custom chassis, but without other proprietary hardware
- Data network interface is via 10/25GbE
- Up to 4 disk modules of 8 HDDs each
- Modules can be quickly swapped via SAS cables

Mark6 - Hardware refresh

MIT HAYSTACK OBSERVATORY

1. Every few years, we need a hardware refresh due to component EoL
2. Last refresh provided the opportunity to explore:
   1. PCIe4.0
   2. AMD EPYC CPUs (with 128 PCIe lanes)
   3. ATTO HBA
   4. Intel network cards



(a) Updated Mark6 interior with AMD EPYC, Supermicro H12SSL motherboard

(b) Mark6 + 2 R2DBEs in hypobaric chamber

1. Using the existing d-plane software (R. Cappallo), demonstrated we can do >16Gbps.
2. 32Gbps operation in two modes was tested: (2x 16Gbps) and (4x 8Gbps)
3. This mode also tested at 4500m equivalent in hypobaric chamber

Mark6(+) Hardware selection

MIT HAYSTACK OBSERVATORY

1. AMD 16-core CPU - (EPYC 7282, 128 lane PCIe 4.0)
2. PCIe 4.0 capable motherboard (Supermicro H12SSL-CT with 64 GB RAM - supports up to 2TB)
3. 1x 100GbE NIC: Intel E810-CQDA2 (QSFP28)
4. 2x 10/25GbE NIC: Intel XXV710-DA2 (SFP+)
5. 2x HBA: Atto H12F0GT 16-Port 6/12Gb/s SAS (SFF-8644 to SFF-8088)
6. 4x Mark6 disk modules (8 drives each) - Seagate Exos16 16TB HDD
7. Additional high speed cooling fans and custom airflow ducts

Figure: Interior of Mark6+ with 100GbE NIC

But what about a software refresh?

MIT HAYSTACK OBSERVATORY

Several motivating factors for a software refresh:

1. Need to move to a modern OS (Debian Squeeze and CentOS7 are both EoL)
    1. Ubuntu 22.04 LTS selected - support to 2034 (FIPS option available)
2. Need to support new hardware and 100GbE:
    1. ngEHT - future backend (2x 32Gbps VDIF threads on a single 100Gbe interface)
    2. DBEv5 - support for a direct 100Gbe connection (no switch)
3. Need to demonstrate continuous 32Gbps operation of a Mark6 on a single 100Gbe interface.
4. Optionally – demonstrate RX of multiple VDIF threads on a single interface
5. Opportunity to explore newer technologies - particularly kernel bypass packet capture
    1. PF_RING is used by pre-existing software, but various other technologies are available
    2. DPDK looks like the leading candidate for this task
    3. As specialized hardware not needed, and its appropriate for unidirectional UDP capture
6. Need to upgrade cplane to python3

PF_RING vs. DPDK

MIT HAYSTACK OBSERVATORY

1. Existing Mark6 dplane software uses PF_RING as packet capture library
2. Zero-copy mode requires (per-MAC) paid-license, so not utilized
3. Must manage and tune cpu-interrupts for proper performance! – This can be quite a chore when CPUs (frequently) go EoL
4. More limited set of supported devices
5. Device still managed/visible to kernel network stack
6. Not required, but generally operated in promiscuous mode

1. Moving to DPDK allows for:
2. Use of poll-mode drivers - removes the need of cpu interrupts entirely.
3. Takes full control of the network device away from the kernel
4. Buffered/burst packet capture with DMA from device to host-memory
5. Zero-copy packet manipulation (no license fee)
6. However: No kernel-based network management is possible (e.g. no ARP/ICMP support).
7. Ok, since we are treating it as a purely a point-to-point link
8. Also not required, but we operate in promiscuous mode

Updated dplane architecture

MIT HAYSTACK OBSERVATORY

1. Old architecture revolved around packet ring-buffers (per device)
2. New packet processing relies on DPDK mbufs pool
3. mbufs are pre-allocated – must enable hugepages in kernel params
4. mbufs are filled on device burst-read, and free'd back to the pool on last use
5. Instead of ring buffer, data is processed in chunks by various thread pools, and passed from task-to-task via locking stacks/queues
6. With appropriately sized work items, lock contention is a non-issue
7. For the most part, in order to simplify and preserve packet-ordering on reception, some thread-pools have size 1 (single threaded):
    1. The packet receiver – one per interface
    2. The VDIF thread ID sorter – one per interface (needed when multiple VDIF threads present on a single interface)
    3. The scatter-gather block constructor (one per interface)
8. Exception: The thread pool for SG block writers has size N_{files}

Packet life cycle

MIT HAYSTACK OBSERVATORY

1. Packets are received by NIC
2. Packets are burst read into mbufs (extracted from pool) by poll-mode driver call
3. Packet burst is passed to VDIF thread ID sorter, packets are sorted and passed to block constructor
4. Each VDIF thread is assigned a scatter-gather block constructor, which collects packets until it is full
5. Completed blocks are passed to waiting writers, which write to next available HDD
6. Empty blocks are recycled back to the collectors, and used packets to the mbuf pool

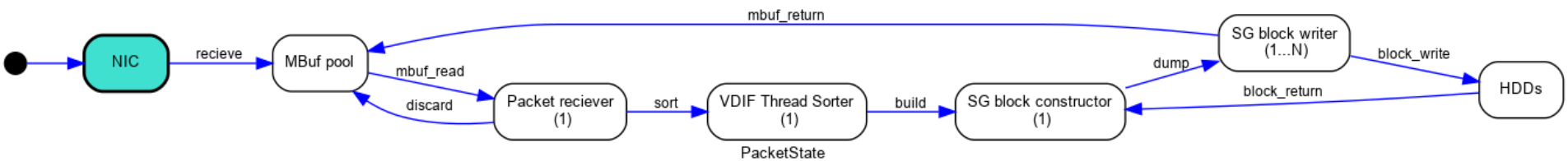


Figure: Packet mbuf cycle.

Testing/Simulation

MIT HAYSTACK OBSERVATORY

1. Currently don't have a readily available back-end to serve data at required rates
2. Need a dummy data generator (vdif headers + junk payload) on the cheap
3. DPDK packet burst functionality + nanosleep to gets us to an approximate aggregate data rate
4. Can then feed the 100GbE NIC back to itself
5. After packet capture, we gather the scatter-gather files and check VDIF header validity with the tool `dqa`.

Loop-back testing

MIT HAYSTACK OBSERVATORY

Using VDIF packet simulator we tried the following configurations:

1. 1x VDIF thread @ 32 Gbps - continuous, 0 dropped ✔
2. 2x VDIF threads @ 64 Gbps (1 thread recorded) - continuous, 0 dropped ✔
3. 1x VDIF thread @ 59 Gbps - continuous, (dropped <6e-05) ✔
4. 1x VDIF thread @ 64 Gbps 75-80s before memory buffer* exhausted - burst mode only
5. 2x VDIF threads @ 64 Gbps 20-30s before memory buffer* exhausted - burst mode only

Figure: Back of Mark6 - 100GbE fiber in yellow

- This testing was done with single actuator drives with an individual write speed of 261MB/s
- Results suggest we are achieving about 85-90% of the theoretical aggregate write throughput
- Latest available HDDs have advertised max sustained write speeds up to 285MB/s which might enable us to go a recording rate as high as 64Gbps continuous without resorting to SSDs or multi-actuator drives
- *memory buffer size was 36GB

Some ugly details and dead-ends

MIT HAYSTACK OBSERVATORY

1. The kernel parameter isolcpus is necessary to keep the cores hosting the packet-receive thread and VDIF sorter thread from handling other tasks
2. Write balancing is tricky - disk performance varies. Forcing equal writes across disks limits the speed to the slowest performer.
3. Alternative: (first-come-first-served) results in $\sim 1\%$ size variance
4. DPDK mbuf buffer pool size must be $2^M \times \mathrm{MTU}$ (which limits RAM buffer configurability)
5. When the RAM buffer is full, data must be dropped.
6. Time-slip between packet simulator and system makes long schedules (24hrs) difficult to test.
7. Some failed experiments:
   1. Async `io_uring` seemed like a good idea, but was found to be slower than standard fwrite
   2. Likewise, using unbuffered system call to 'write' (not fwrite), requires data to be packaged in 4K blocks
   3. However that doesn't work with the VDIF packet size (8192+32), and since mbufs are not guaranteed to be contiguous, a copy is needed ✗

Future work/experiments

MIT HAYSTACK OBSERVATORY

Near term goals:

1. Populate the simulated packets with payload data
2. Perform integration and verification testing with ngEHT DBE
3. The new dplane needs to be integrated with the cplane software (some minor API changes expected)

Future work:

1. Examine performance higher speed single-actuator drives and dual actuator HDDs (Seagate MACH.2)
2. A large RAM buffer could possibly obviate the need for continuous recording capability for most practical applications, but would be an interesting space to explore
3. Test 100GbE system performance at high altitude conditions

Acknowledgements

MIT HAYSTACK OBSERVATORY

This work was done with the hardware and support of NASA-SGP, the ngEHT, and the EHT: