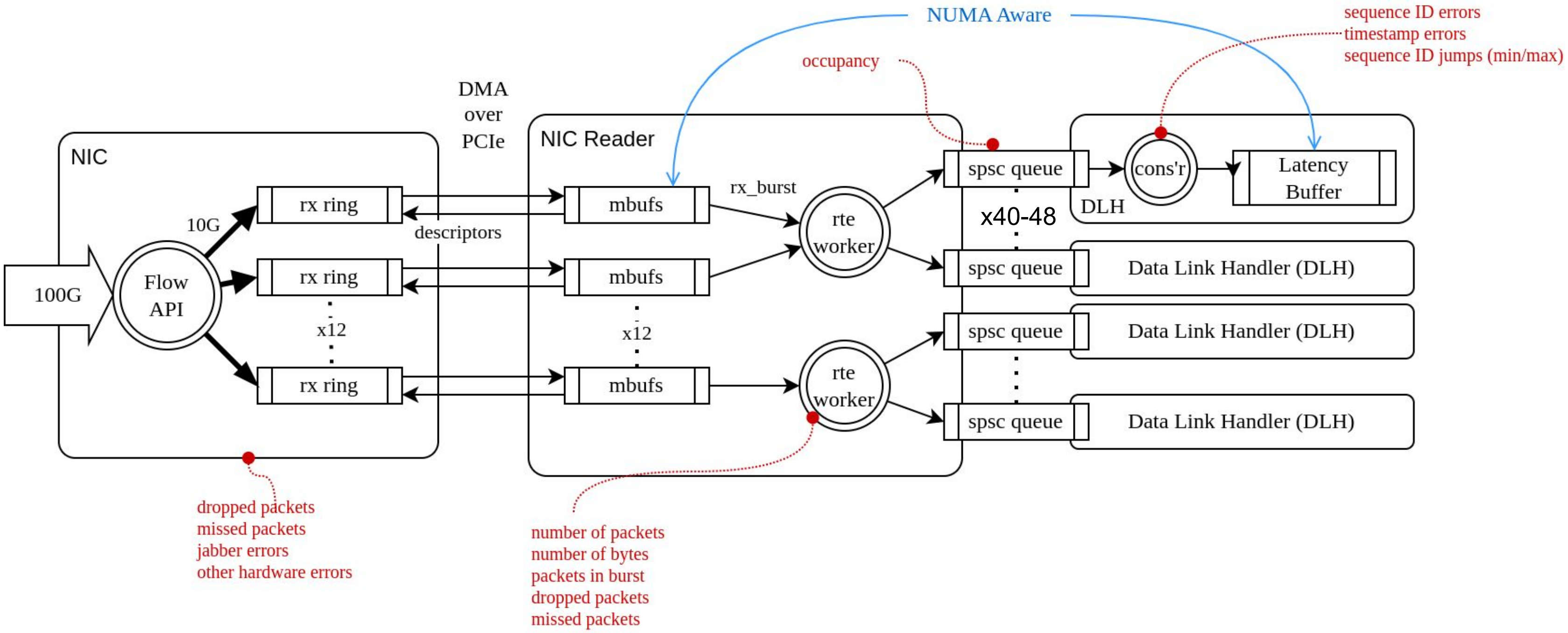
Ethernet readout optimization

Roland Sipos - for the DUNE DAQ
CERN

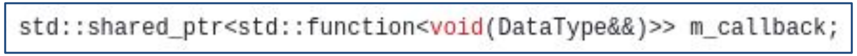
DUNE DAQ CoreSW Meeting
3rd April 2024

Data movement - original



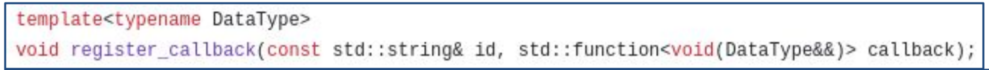
Callback feature in 4.4.0

- Readoutlibs’ DataMoveCallbackRegistry for intra-process readout modules
  - Similar API and structure as IOManager, but holds std::functions with specific signature



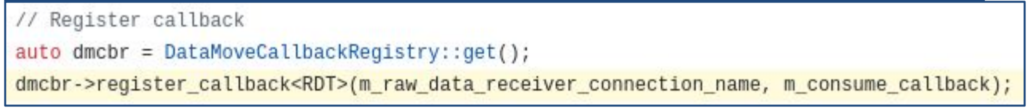
```
std::shared_ptr<std::function<void(DataType&&)>> m_callback;
```

- DataLinkHandler modules advertise/register a callback function, that moves the data to the preprocessing pipeline, and then writing to the LatencyBuffer.
  - Registry:



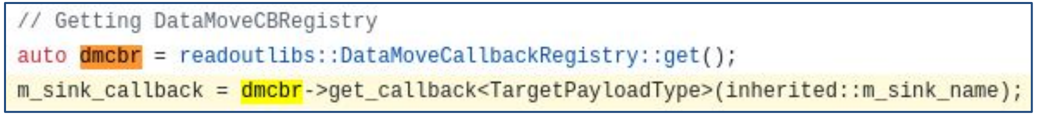
```
template<typename DataType>
void register_callback(const std::string& id, std::function<void(DataType&&)> callback);
```

  - ReadoutModel:



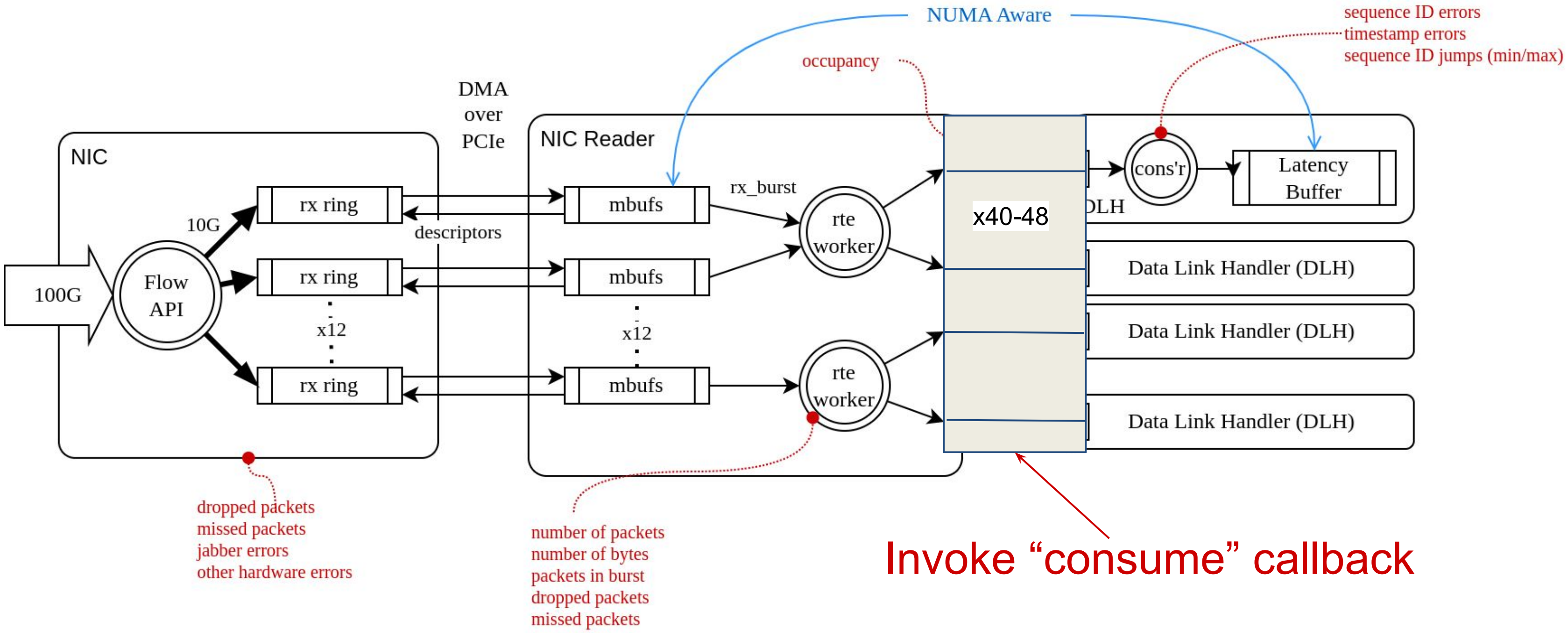
```
// Register callback
auto dmcbr = DataMoveCallbackRegistry::get();
dmcbr->register_callback<RDT>(m_raw_data_receiver_connection_name, m_consume_callback);
```

- I/O card receiver modules (NICReceiver) could look-up which callback functions to invoke to move the data from the producer thread’s context directly to the DLHs’:



```
// Getting DataMoveCBRegistry
auto dmcbr = readoutlibs::DataMoveCallbackRegistry::get();
m_sink_callback = dmcbr->get_callback<TargetPayloadType>(inherited::m_sink_name);
```

Data movement - Callbacks



Invoke “consume” callback

Callback feature comments

- Optional path, inferred from connection name to be enabled
  - Necessary, as we don’t want to break other datatypes (trigger, TPs, etc.)
  - Ugly, “hacky”, I know: If the *raw_input* connection has a “*cb_*” prefix

- Would it result with any perf. gain for TPs to TPDataLinkHandlers!?
  - Maybe, but probably callbacks’ invocation needs parallel async techniques either on the producer or consume callback owner class. (Tricky, not SPSC.)

- Forward port to “develop”?
  - Feedback for appfwk review first? Or that’s too far ahead, seeing the gain?
  - Improvements and common API/utility promotion are possible for sure…
    - Generalize for network: RPC variant?
    - GenericCallback instead of move signature only functions?

Resource utilization reduction

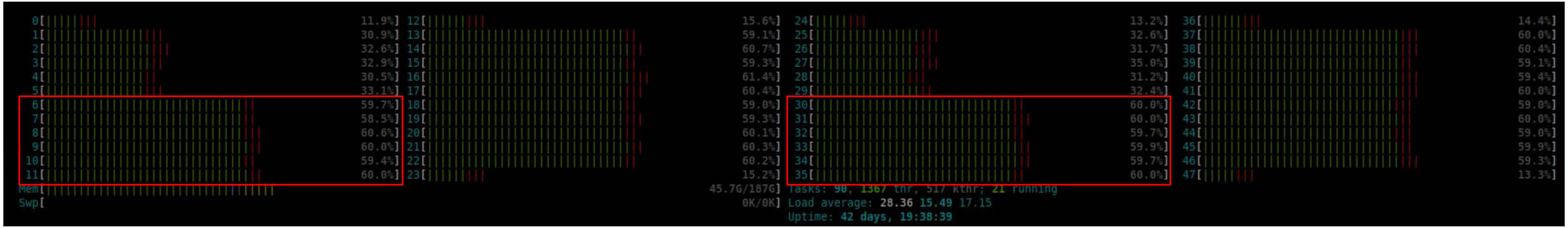
- Removes one full copy (intermediate buffering) of the raw data stream:
  - Eliminates consumer threads: no resource utilization due to polling
  - Eliminates context switching between prod/consumers
  - LLC and first level cache miss reduction due to first two points

- Memory bandwidth utilization percentage difference: **~37% less**!
  - Can be utilized as natural headroom for SNB recording!

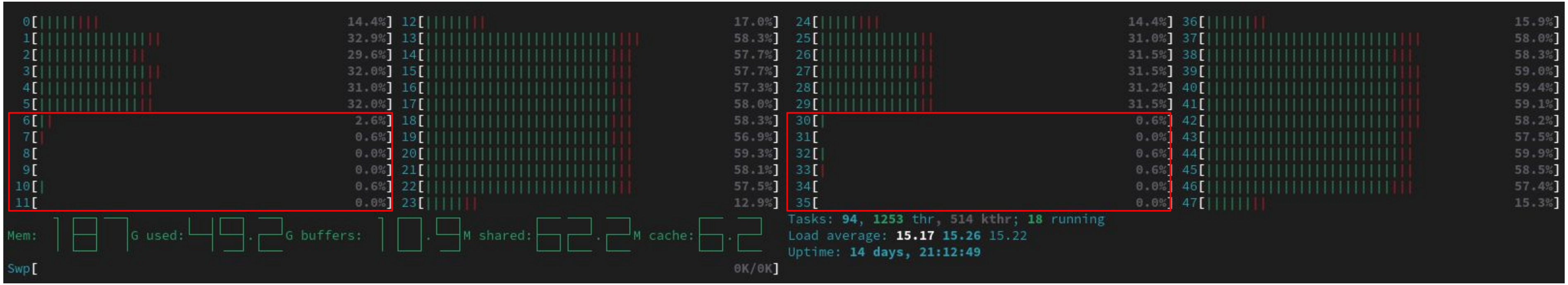
- CPU utilization reduction: **12 vCores (Phys & their HT) x ~60%**
  - This one is on Intel Xeon Gold 5118 @ 2.3 GHz (Skylake)
  - Heavily depends on CPU!
  - But even on newer generations we see ~10 vCores x ~35% reduction (CPUs with more cores and higher clocks)

CPU utilization

- **Non negligible on CPUs with lower core counts and clocks!**
- On Intel Xeon Gold 5118 @ 2.3 GHz (Skylake) -> before/after callbacks



```
  0[   11.9%] 12[   15.6%]  24[   13.2%]  36[   14.4%]
  1[   30.9%] 13[   59.1%]  25[   32.6%]  37[   60.0%]
  2[   32.6%] 14[   60.7%]  26[   31.7%]  38[   60.4%]
  3[   32.9%] 15[   59.3%]  27[   35.0%]  39[   59.1%]
  4[   30.5%] 16[   61.4%]  28[   31.2%]  40[   59.4%]
  5[   33.1%] 17[   60.4%]  29[   32.4%]  41[   60.0%]
  6[   59.7%] 18[   59.0%]  30[   60.0%]  42[   59.0%]
  7[   58.5%] 19[   59.3%]  31[   60.0%]  43[   60.0%]
  8[   60.6%] 20[   60.1%]  32[   59.7%]  44[   59.0%]
  9[   60.0%] 21[   60.3%]  33[   59.9%]  45[   59.9%]
 10[   59.4%] 22[   60.2%]  34[   59.7%]  46[   59.3%]
 11[   60.0%] 23[   15.2%]  35[   60.0%]  47[   13.3%]
Mem[              45.7G/187G] Tasks: 90, 1367 thr, 517 kthr; 21 running
Swp[                 0K/0K] Load average: 28.36 15.49 17.15
                             Uptime: 42 days, 19:38:39
```

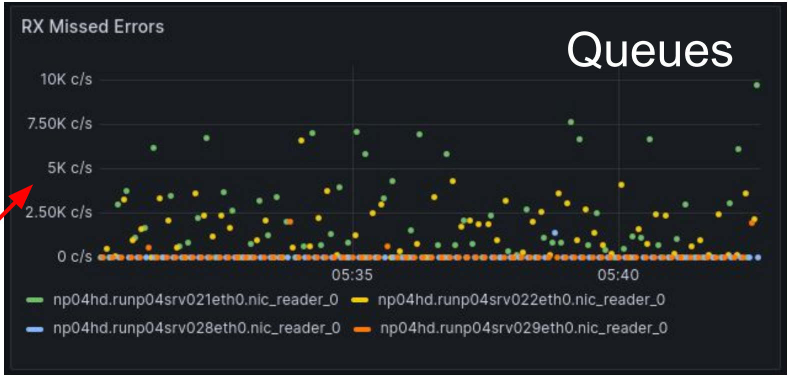


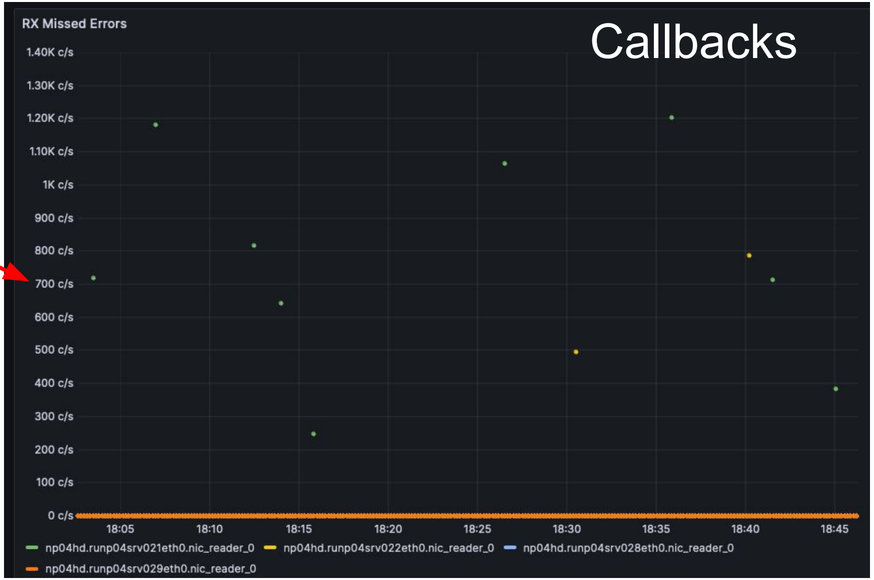
```
  0[   14.4%] 12[   17.0%]  24[   14.4%]  36[   15.9%]
  1[   32.9%] 13[   58.3%]  25[   31.0%]  37[   58.0%]
  2[   29.6%] 14[   57.7%]  26[   31.5%]  38[   58.3%]
  3[   32.0%] 15[   57.7%]  27[   31.5%]  39[   59.0%]
  4[   31.0%] 16[   57.3%]  28[   31.2%]  40[   59.4%]
  5[   32.0%] 17[   58.0%]  29[   31.5%]  41[   59.1%]
  6[    2.6%] 18[   58.3%]  30[    0.6%]  42[   58.2%]
  7[    0.6%] 19[   56.9%]  31[    0.0%]  43[   57.5%]
  8[    0.0%] 20[   59.3%]  32[    0.6%]  44[   59.9%]
  9[    0.0%] 21[   58.1%]  33[    0.6%]  45[   58.5%]
 10[    0.6%] 22[   57.5%]  34[    0.0%]  46[   57.4%]
 11[    0.0%] 23[   12.9%]  35[    0.0%]  47[   15.3%]
                              Tasks: 94, 1253 thr, 514 kthr; 18 running
Mem: 187G used: 49.2G buffers: 10.9M shared: 62.2M cache: 6.2   Load average: 15.17 15.26 15.22
                              Uptime: 14 days, 21:12:49
Swp[                  0K/0K]
```

Missed packets

When the processing (copying out data from the DMA buffers) is lagging. Data is missed!

- Queues: Order of thousands missed packets in every few seconds
- Callbacks: Occasional spikes to a few hundred in every few minutes





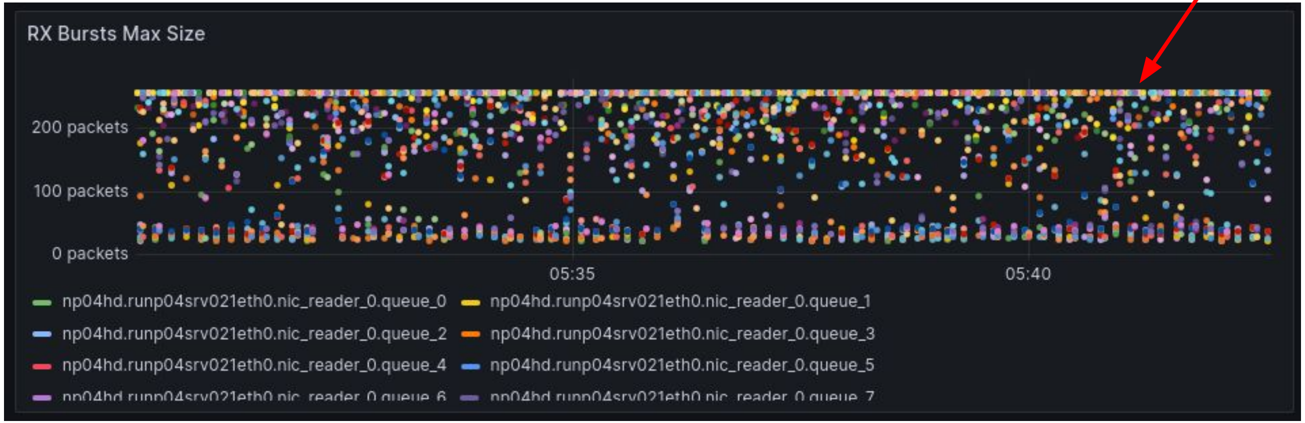
Where are these coming from?
We knew, that we are sensitive to interrupts, but callbacks only won’t help with that, as this comes from the producer thread (in DPDK).

Something is still wrong…

Alessandro’s evaluation and testing

- Alessandro did extensive testing on the AMD server to tweak interface and DMA memory pool configurations in DPDK

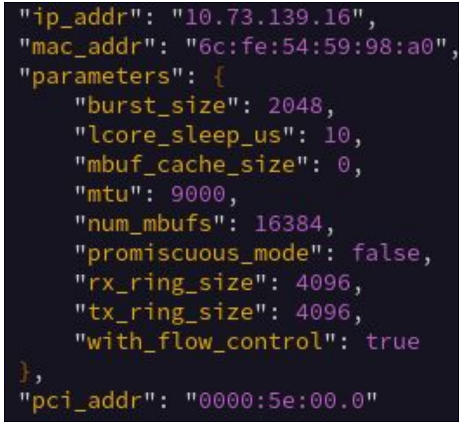
- Main find: The interface polling function (“rx burst”) returns with **full size** very frequently. Our DPDK/iface configuration needs a revisit.



DPDK/iface config revisit

- **Burst size**: maximum number of packets/descriptors to poll.
  This allows a run-to-completion processing stack to statically fix or to dynamically adapt its overall behavior through different global loop policies.
  - **Previous value: 256**
- **Lcore_sleep_us**: Microseconds to sleep if there are no full buffers in the burst.
- **Mbuf_cache_size**: Complicated, we didn't understand this properly…
  For details have a look on the mempool API docs
  - **Previous value: 256**
- **Num_mbufs**: The number of mbufs (DPDK DMA buffer segments) to allocate.
  - **Previous value: 8191**
- **RX/TX ring size**: Number of RX/TX descriptors.
  - That is the mechanism you communicate to the NIC hardware - e.g., in the case of RX, for passing pointers to empty buffers that you pass to NIC and getting pointers for filled buffers that NIC returns.
  - Usually this parameter has hardware limits (max. num. of descriptors).
  - **Previous value: 1024**

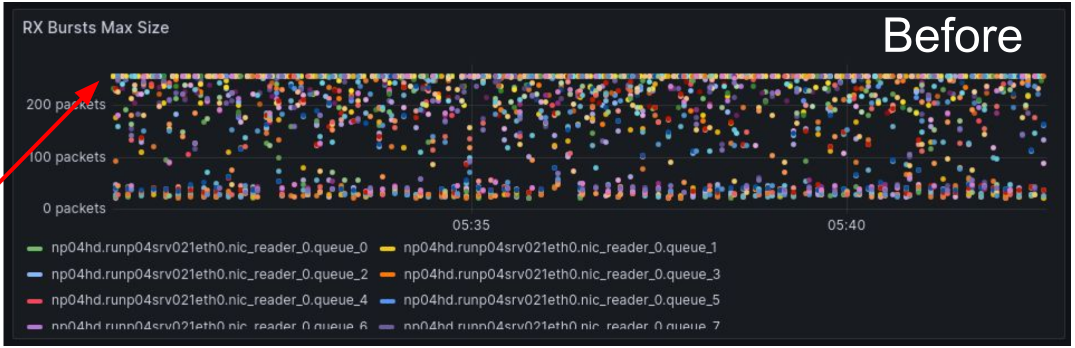
New "golden" config



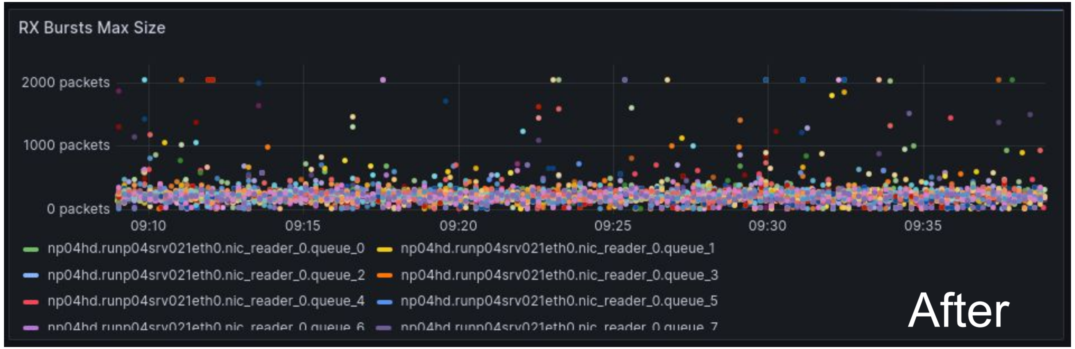
```
"ip_addr": "10.73.139.16",
"mac_addr": "6c:fe:54:59:98:a0",
"parameters": {
    "burst_size": 2048,
    "lcore_sleep_us": 10,
    "mbuf_cache_size": 0,
    "mtu": 9000,
    "num_mbufs": 16384,
    "promiscuous_mode": false,
    "rx_ring_size": 4096,
    "tx_ring_size": 4096,
    "with_flow_control": true
},
"pci_addr": "0000:5e:00.0"
```

Iface config tuning

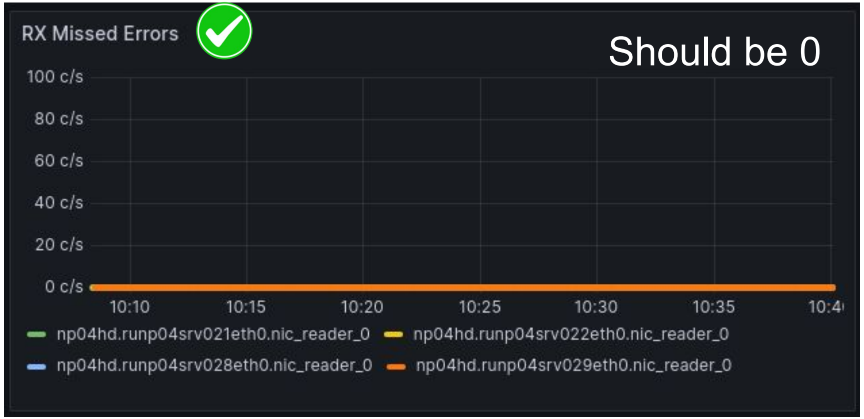
- Strategy for our traffic characteristics is to poll (sensitive to interrupts) as many packets as we can in one go **(burst)**. Basically to reduce the impact of interrupts and kernel events.
  - Increase num. descriptors
  - Allocate more mbufs
  - Increase burst size



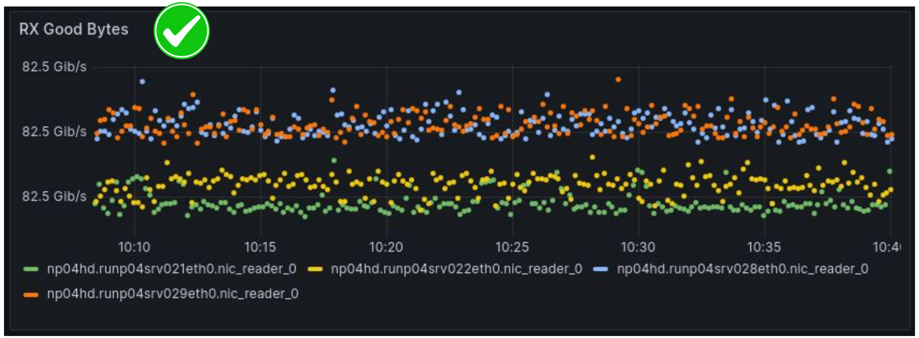
Take note of the removed linear roof: Reduced full bursts

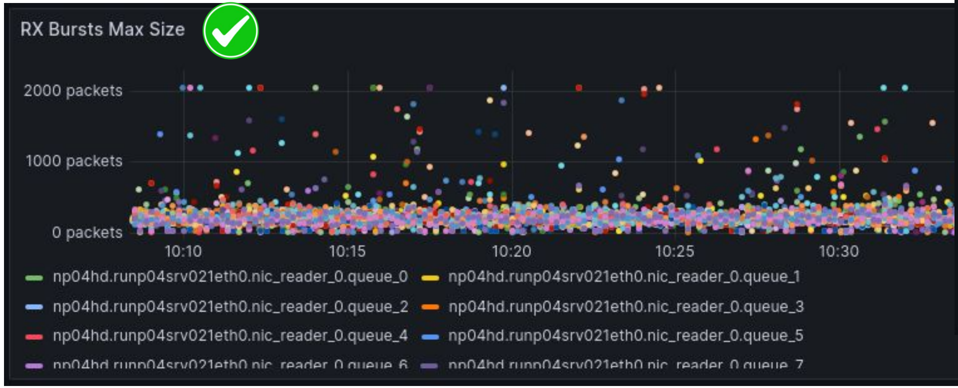


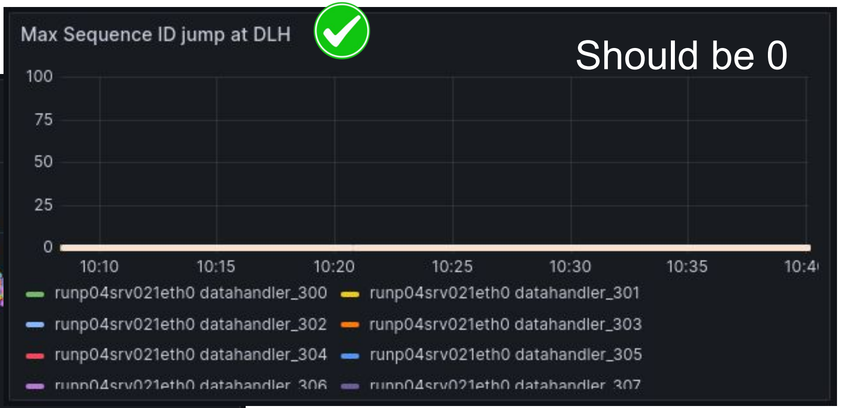
New performance


RX Missed Errors

Should be 0


RX Good Bytes


RX Bursts Max Size


Max Sequence ID jump at DLH

Should be 0

Outlook

- There are still some very rare occurrences of missed packets on 021. (That machine has some kernel parameters under testing, might be related!)

- Kernel isolation should still be pursued, as my suspicion is that will still help the AMD due to the higher I/O die latency on the processor

- This optimization builds up confidence in moving on to scalability aspects
  - 2 NICs in the same server

- Also, SNB recording evaluation and improvements can continue

- Topology evaluation opens up again
  - Symmetric or asymmetric configuration of I/O devices and processing threads

Summary

- Ethernet readout with TPG, resource headroom available (but no SNB recording) on a dual-socket server with CPUs that are from Q3 of 2017 (mid-range Skylake)

- Great work and sprint from the perf-testing activity team for fddaq-4.4.0!
  - Building up experience and confidence in understanding system characteristics and requirements

- The path ahead is clear what we aim to test in our upcoming sprint
  - Many other things to test and sort out, but we managed to breach through a showstopper problem

- Questions and comments are welcome!