RDMA DEBUG AND DIAGNOSTICS

Jason Gunthorpe

Mellanox Technologies

[ March, 2019 ]
- Debug information and techniques are spread all over the place
- The design is undergoing some change and consolidation
- Survey where we are and where things are going

- Monitor
- Inspect
- Debug
Only IB/OPA have standard central fabric managers

- IB fabrics can use ‘ibdiags’ tools
- On node data can be accessed from central management

Ethernet protocols rely on Ethernet tools

- Switch based software for fabric inspection, some times fabric wide
- SNMP and other tools for node based data collection
- RDMA related information is kind of on the side and not brought into existing managment

All protocols share a fairly similar API for on-node debug

- But many things are driver specific, so lots of advanced features are not broadly available
COUNTERS
• Counters are one of the best tools for monitoring and problem analysis
  - Can deal with high data rates
  - Un-intrusive
  - Error counters un-ambiguously indicate a problem
  - Give broad insight into what is happening

• Can be viewed using different tools, Ethernet based NICs have a mixture of ethernet based and RDMA focused tools

• Global and per-object counters are available

• Quite fragmented now
Basic always available counters. Defined by IETF as part of the SNMP MIB for Ethernet

Basic link counters come from ‘ip -s link show’ - these include RDMA traffic

2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
tmode DEFAULT group default qlen 1000

link/ether d8:9e:f3:17:05:93 brd ff:ff:ff:ff:ff:ff
RX: bytes packets errors dropped overrun mcast
4192125393 4294042 0 0 0 83046
TX: bytes packets errors dropped carrier collsns
705979348 2876860 0 0 0 0
‘ethtool -S’ shows counters from the Ethernet driver. Each driver has a unique set of counters. Consult driver documentation for details:

NIC statistics:

- rx_packets: 0
- rx_bytes: 0
- tx_packets: 16
- tx_bytes: 2316
- tx_tso_packets: 0
- tx_tso_bytes: 0
- tx_tso_inner_packets: 0
- tx_tso_inner_bytes: 0
- tx_added_vlan_packets: 0
- tx_nop: 0
- rx_lro_packets: 0
- rx_lro_bytes: 0
...

...
IB PORT COUNTERS

- Classic perfquery
- Same counters are available in /sys/class/infiniband/XXX/ports/1/counters
- Definitions are found in the InfiniBand Architecture

```bash
# Port counters: Lid 13397 port 1 (CapMask: 0x00)
PortSelect:.........................1
CounterSelect:......................0x0101
SymbolErrorCounter:...............0
LinkErrorRecoveryCounter:........0
LinkDownedCounter:...............0
PortRcvErrors:....................0
PortRcvRemotePhysicalErrors:....0
PortRcvSwitchRelayErrors:........0
PortXmitDiscards:..................0
PortXmitConstraintErrors:........0
PortRcvConstraintErrors:.........0
CounterSelect2:....................0x00
LocalLinkIntegrityErrors:........0
ExcessiveBufferOverrunErrors:....0
QP1Dropped:.......................0
VL15Dropped:......................0
PortXmitData:......................563
PortRcvData:......................0
PortXmitPkts:......................16
PortRcvPkts:.......................0
PortXmitWait:.....................0
```
RDMA DRIVER COUNTERS

- **Driver specific counters are available in the `hw_counters` directory**
  
  ```
  hw_counters/rp_cnp_ignored:0
  hw_counters/resp_local_length_error:0
  hw_counters/np_ecn_marked_roce_packets:0
  hw_counters/req_remote_invalid_request:0
  hw_counters/local_ack_timeout_err:0
  hw_counters/lifespan:12
  hw_counters/req_cqe_error:0
  hw_counters/rnr_nak_retry_err:0
  hw_counters/np_cnp_sent:0
  hw_counters/rp_cnp_handled:0
  hw_counters/implied_nak_seq_err:0
  hw_counters/req_cqe_flush_error:0
  hw_counters/packet_seq_err:0
  hw_counters/duplicate_request:0
  hw_counters/out_of_buffer:0
  ...
  ```

- **Plans to expose these via a new ‘rdma statistics’ command**
• Upcoming ‘rdma statistics’ command

• Goal to consolidate all device and port counters under one command
  • RDMA interesting ethtool counters
  • perfquery counters for the port
  • Driver specific RDMA counters from sysfs
RDMA ON DEMAND COUNTERS

- Still in development
- Isolate objects and then count things on them
- Request a counter set for a single QP:
  
  $ \texttt{rdma statistic bind link XXX lqpn 1234}$
  
  $\texttt{rdma statistic show qp}$

- Also some thinking on providing per-process counters and other groupings
- What counters are available is up to the driver

- ‘\texttt{rdma statistics}’ should eventually show all the counters related to RDMA and the device that are currently spread about
RDMA OBJECTS

- Now have visibility into RDMA objects: Device, PD, MR, QP, CM_ID, CQ and contexts via the ‘rdma’ tool
  - Shows objects created by the kernel
  - Shows which user space process created the object

- Various ways to search and display the objects

- device/link/port can be seen in multiple ways
  - For RoCE ‘ip link’ will show the underlying ethernet device
  - ‘rdma link’ will show if the kernel RDMA device is present
  - ‘ibv_devinfo’ will show if the device is visible to user verbs

- rdma is now the way to learn what netdev is connected to which IB device
device/link/port can be seen in multiple ways

- For RoCE `ip link` will show the underlying ethernet device
  IP information is relevant to RoCEv2
- `rdma link` will show if the kernel RDMA device is present
- `ibv_devinfo` will show if the device is visible to user verbs

Information like what the physical link is doing is still scattered

`rdma` is now the way to learn what netdev is connected to which IB device
Per-object data, are available (varies depending on driver) via ‘rdma resource show XX -d’

dev hnseth0 cqe 1023 users 2 poll-ctx WORKQUEUE pid 0 comm [ib_core] drv_state 2
drv_ceqn 0 drv_cqn 0 drv_hopnum 1 drv_pi 0 drv_ci 0 drv_coalesce 0 drv_period 0
drv_cnt 0

This capability is new, drivers are starting to add these reports
More objects and more data are coming to this interface
Most useful for driver debugging
For netstack we’d traditionally use ‘ss’, ‘netstat -a’ and ‘lsof’ to see information about active connections

In RDMA we now can use the rdma command to list this information

Latest work is exposing PID data as well allowing ‘lsof’ like functionality for RDMA

CM_ID alone is a ‘listening socket’
CM_ID with a QP is an ‘established socket’
USE KERNEL DEBUGGING

- Turn on dev_dbg messages in the core and driver:

- Use kernel ftrace
  https://lwn.net/Articles/365835/

- Can give insight into what is going on with the objects
  - ie many drivers have debugging prints when uverbs calls are done wrong
  - Driver behavior here is not standardized
PERFORMANCE
MEASUREMENT

- Usually done to test or stress the system
- In ethernet we’d use something like iperf3/4, netperf, etc
- Various specific RDMA tools, ib_rdma_bw, ib_rdma_lat, RDMA perftest attempt to measure and stress
- Many applications have their own stress testing tools as well

- Combine with counter monitoring to make sure no negative events occur
Obtaining packet traces can help understand what could be wrong in some cases

Native ethernet would use tcpdump/wireshark

Doesn’t always work for RoCE traffic
  • Support via libpcap for devices that implement sniffing via verbs flow steering

Instead
  • Driver specific command for dumping (eg ibdump on Mellanox)
  • Configure switches to mirror traffic to another NIC and use tcpdump/etc
  • Optical splitting

Generally hard, and falls down at higher speeds.

Most useful for protocol debugging these days
**Overview**

**Inspection**
- `ethtool / perfquery`
- `ip link`
- `rdma`
- `lldptool / dcbtool`
- `ss`
- `ibv_devinfo`

**Performance**
- `iperf3`
- `rdma_bw`
- `RDMA perftest`
- `tcpdump / wireshark`
When debugging, exercise care!
- Formulate a hypothesis
- Change ONE variable
- Disprove hypothesis?
- Change ONE more variable, repeat

Gather multiple points of evidence to support the hypothesis
Correlation != Causation

Having the wrong idea / wrong description what the problem is makes it very difficult to fix
THANK YOU
Jason Gunthorpe
Mellanox Technologies