14th ANNUAL WORKSHOP 2018

HOT UNPLUG SUPPORT FOR RDMA DEVICES

Matan Barak, SW Architect
Mellanox Technologies LTD.

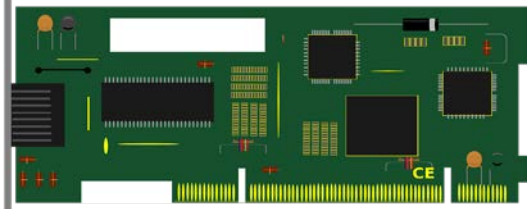
[April, 2018]

AGENDA

- **Introduction**
  - Hot plug and unplug events
  - What is wrong with today's mechanism?
  - Solution components
- **Notifications for hot plug/unplug events**
- **Device list changes dynamically**
- **User-space interaction**
  - Refreshing device list
  - Hot unplug on an opened device
- **Disassociate context**
- **Latest changes in user-space libs**
- **Challenges**
  - ioctl() based commands
  - Emulating completions
  - librdmacm

HOT PLUG AND UNPLUG EVENTS

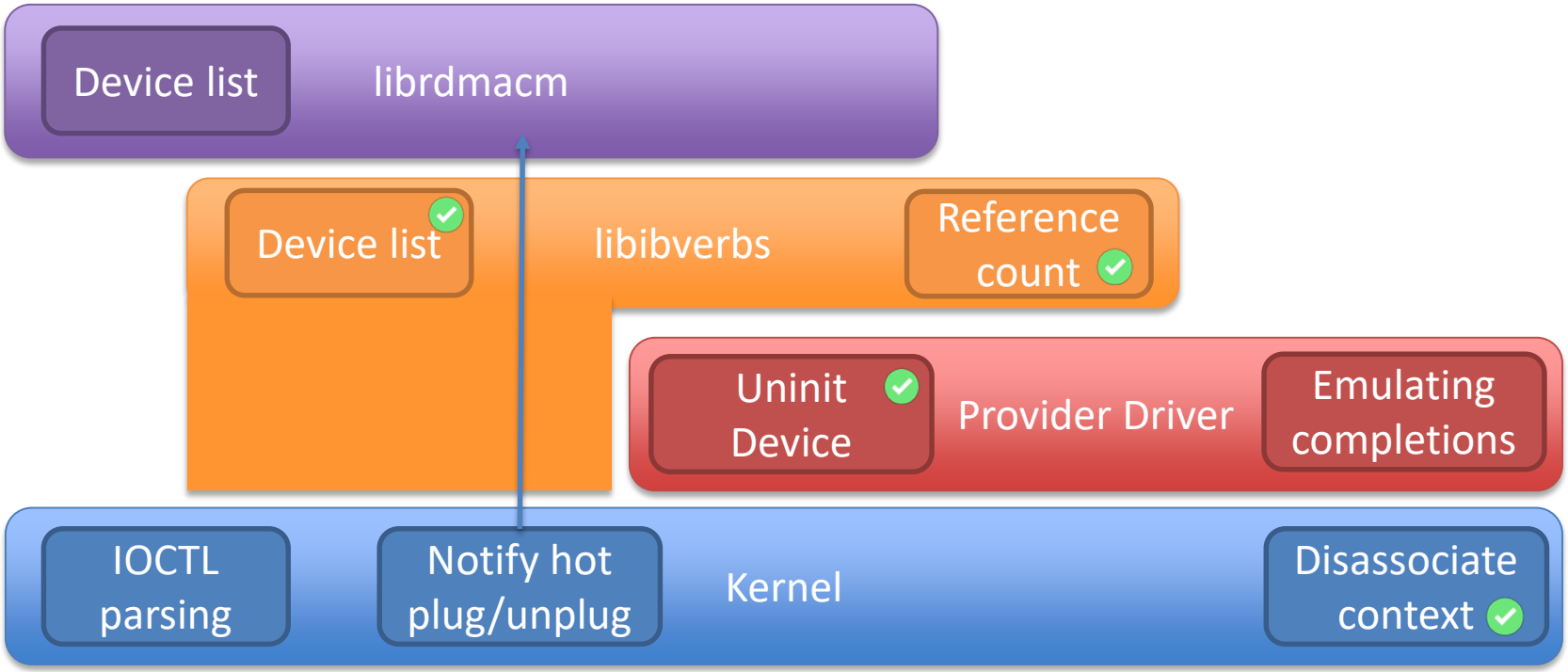
- **Hot plug/unplug events**
  - Physically inserting/removing a PCI device
  - Hot plug/unplug a device to a VM
  - Add/Remove an IB device driver
  - Hardware receives a fatal event



- **Dealing currently with hot-unplug**
  - Application gets an IB_EVENT_DEVICE_FATAL in its async event channel fd
  - Application tries to close all resources
    - It gets a –EIO error from the kernel → Resource leak
  - Closing the context
  - No completion events, QP is getting full or failing with immediate errors.
  - Rdma-cm sends RDMA_CM_EVENT_DEVICE_REMOVAL on all opened IDs

- **Dealing currently with hot-plug**
  - User-space doesn't have a well-established way to get this event
    - If it had, It should have re-scanned all IB devices

SOLUTION COMPONENTS



= Implemented

(*) In production in one of the biggest clouds

NOTIFY FOR DEVICE CHANGES

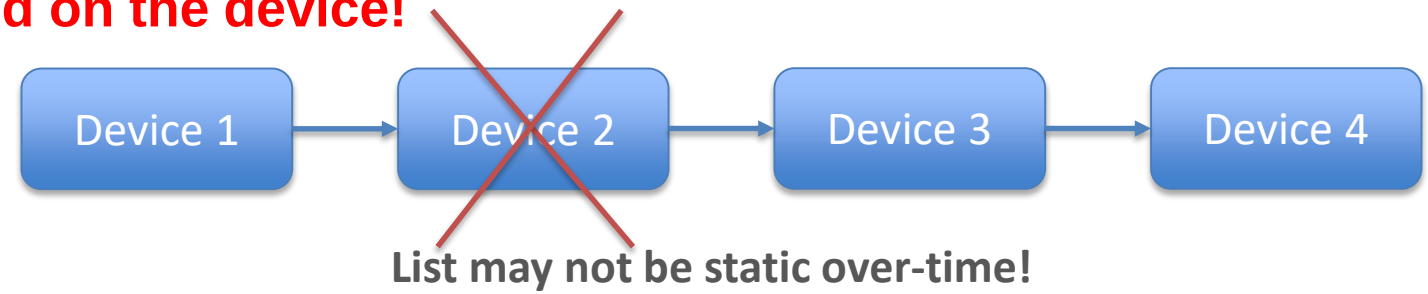
- **Why do we need to be notified?**
  - Need to know that a device we're currently working on is dying.
  - A better device was plugged and we want to move to the new device.
  - We might want to use another device in the future.
  - Moving a device between VMs according to the overload [MSFT]

- **Notifications mechanism**
  - Application has an active context on a device
    - It gets an `IB_EVENT_DEVICE_FATAL` in its async event channel fd
  - No active context on a device
    - Currently no mechanism is implemented, several possibilities:
      - Filtering events from netlink socket on NETLINK_KOBJECT_UEVENT group
      - Filtering udev events via libudev
      - INotify (filesystem) events ("create" + "delete" on '/dev/infiniband)
      - New libibverbs (uverbs) fd channel – maybe part of librdmacm?

DEVICE LIST

- **Opening an IB device requires scanning the list of devices**
- **libibverbs scans the device list via sysfs when `ibv_get_device_list()` is called**
- **A device is then chosen either by matching its name or GUID**
- **librdmacm scans the device list only when it initializes.**
- **New in 2017: Subsequent calls to `ibv_get_device_list()` refresh the list by adding new devices and deleting plugged out devices.**
  - Scanning is done by name and file's timestamp.

Unfortunately, new devices could be plugged and old devices could be unplugged. This could happen even when a context is opened on the device!



REFRESHING THE DEVICE LIST

Application uses libibverbs new notification event fd channel (or a generic netlink socket layer) by calling ibv_open_sys_event_channel()

It's recommended to change the event channel to non-blocking mode.

Get pending events by calling
ibv_get_sys_event(struct ibv_sys_event* event, size_t sz)

Keep pooling for events until –EAGAIN (no events exist)

Wait on the fd until HOT_PLUG/HOT_UNPLUG event arrives

Call `ibv_get_device_list()` to refresh device list

HOT UNPLUG ON AN OPENED DEVICE

Kernel has disassociated the context and destroyed all objects

Application gets an `IBV_EVENT_DEVICE_FATAL` on the async FD

Application closes all its resources

Destroy resources should succeed
(but currently it'll return `-EIO` ☹)

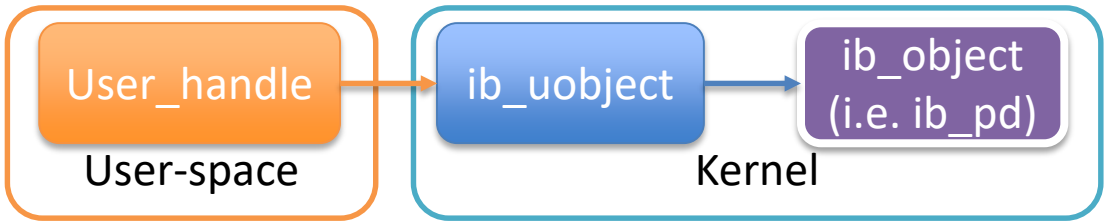
Trying to create/modify/query a resource will result in an error code from the kernel

Close the context with `ibv_close_device()`

DISASSOCIATE CONTEXT

- The device driver notifies that it's being unloaded or unplugged.
- The kernel destroys all IB objects and currently their handles part in the kernel (ib_uobject).
- If the user polled the async event fd, it's being woken up.
- Calling the kernel result in –EIO for every command.

Current implementation lacking



Problems

- Only mlx4 and mlx5 based devices implements disassociate context.
- The kernel destroys all its user-space object handles (ib_uobject) and returns a failure for every command, rather than keeping these handles and successfully destroying them in DESTROY_XXXX verbs.

LATEST CHANGES IN USER-SPACE LIBS

- **`ibv_get_device_list()` creates a new ibv_device list of the current snapshot**
  - Re-read the sysfs
  - Finds matching drivers
  - Creates new verbs_devices and removes unplugged ones
- **Add an "`uninit_device()`" to libibverbs←→provider driver interface. After this function returns, no more references to verbs_device(s) are allowed.**
  - The provider should free any memory it allocated for the verbs_device in this call.

NEW IN 2017

- **Verbs_device will encompass refcnt**
  - Increased by `ibv_get_device_list()`/`ibv_open_device()`
  - Decreased by `ibv_close_device()`/`ibv_free_device_list()`
    - libibverbs frees verbs_device (calling `uninit_device()`) when the refcnt is down to zero.
- **Unplug requires the application to close all IB resources on the ibv_context.**
- **We still lack an event to update the device list.**

CHALLENGES – IOCTL() BASED COMMANDS

Problem

- **Each kernel provider driver could have its own objects, methods and attributes, without affecting the common code.**
- **Implemented by passing provider specific information and function pointers to the generic parser.**
- **In hot unplug, we unload the driver with all this information.**

Possible solution

- **We observe that parsing and dispatching is part of the infrastructure.**
- **Dynamically allocate parsing tree (parsing guidelines) by infrastructure.**
- **No provider specific destroy methods.**
  - When driver is unloaded, destroy all actual IB objects (QP, CQ, etc), but keep the kernel user-space representation (ib_uobject).
- **When getting a destroy call, only release the user-space representation in a generic way.**
- **All other methods should return an error as the parsing tree doesn't exist.**

EMULATING COMPLETIONS

Problem

- **When an active device is hot unplugged, there might be some work requests that weren't processed.**
- **While not mandatory by IBTA specification, ideally application should get an `IBV_WC_WR_FLUSH_ERR` completion for each post WR.**
- **Emulating completions in kernel requires kernel driver to be resident (provider user-space driver interacts with its kernel counterpart directly).**

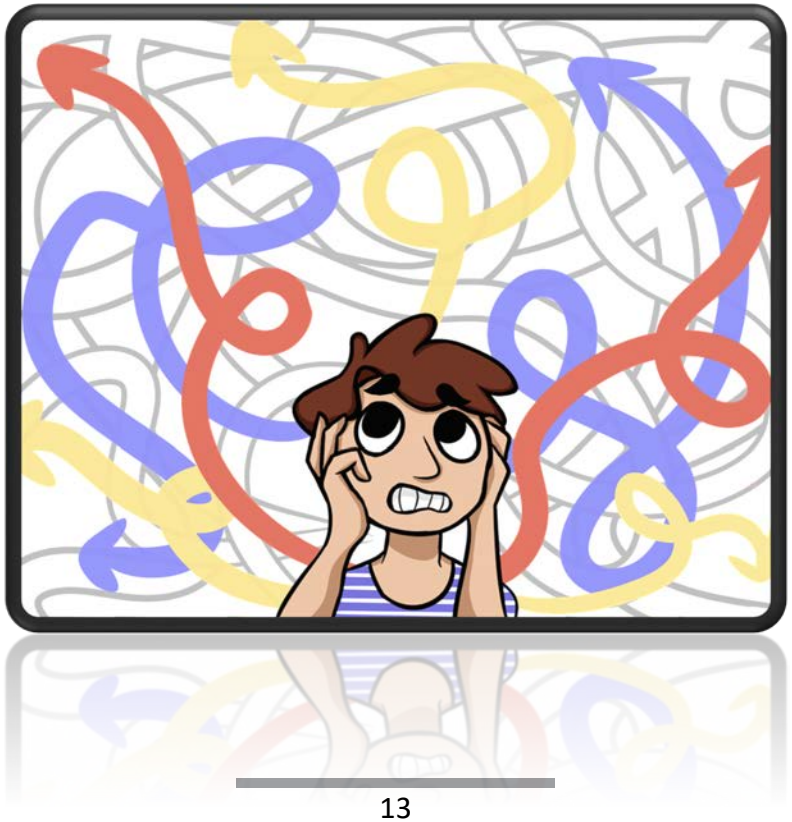
Possible solution

Driver specific solution – for example:

- **Kernel write a "`DEVICE_UNPLUG`" bit in the CQ and wakes up associated event fds.**
- **Provider driver delegate the problem to the user-space driver and detaches.**
- **When the user polls the CQ, this bit is being polled too.**
  - If this bit is set, poll_cq returns a `IBV_WC_WR_FLUSH_ERR` completion after the CQ is empty of real completions.
  - Need to know how many completions you should emulate.
- **Newly posted WR could either fail immediately or create new `IBV_WC_WR_FLUSH_ERR` completions.**

LIBRDMACM

- **RDMA-CM (librdmacm) needs some extra work to support hot plug and unplug.**
  - Maintains a single device list at startup.
  - Need to listen to HOT_PLUG and HOT_UNPLUG events and refresh its device list.



14th ANNUAL WORKSHOP 2018

THANK YOU

Matan Barak, SW Architect

Mellanox Technologies LTD.

Mellanox TECHNOLOGIES