OPENFABRICS ALLIANCE

Extending Verbs API

Tzahi Oved

Mellanox Technologies

April 2013

Agenda

- Introduction
- Why Extensions?
- Backward/Forward Compatibility Factors
- Stack Interfaces Map
- Possible Extension Schemes
- How – API Calls
- How – Provider registration
- How –Structures
- Example – XRC
- Conclusion

Introduction

- Verbs API Evolves to OS-Bypass RDMA standard API
  - Many applications are developed using Verbs
  - Needs to become richer and richer
    - New NIC features are keep being added
  - Must not break existing apps
- Verbs API library vs. vendor provider library
  - May be installed separately
  - May be deployed as part of distro inbox package
  - Must be independent

Why Extensions?

- Enable adding new functionality
  - Introduce new features with new API calls
  - Extend existing APIs
- Uncouple API library (libibverbs) and provider library (libmlx4)
- Unmodified application support
  - App binary backward and forward compatibility
  - App source backward and forward compatibility

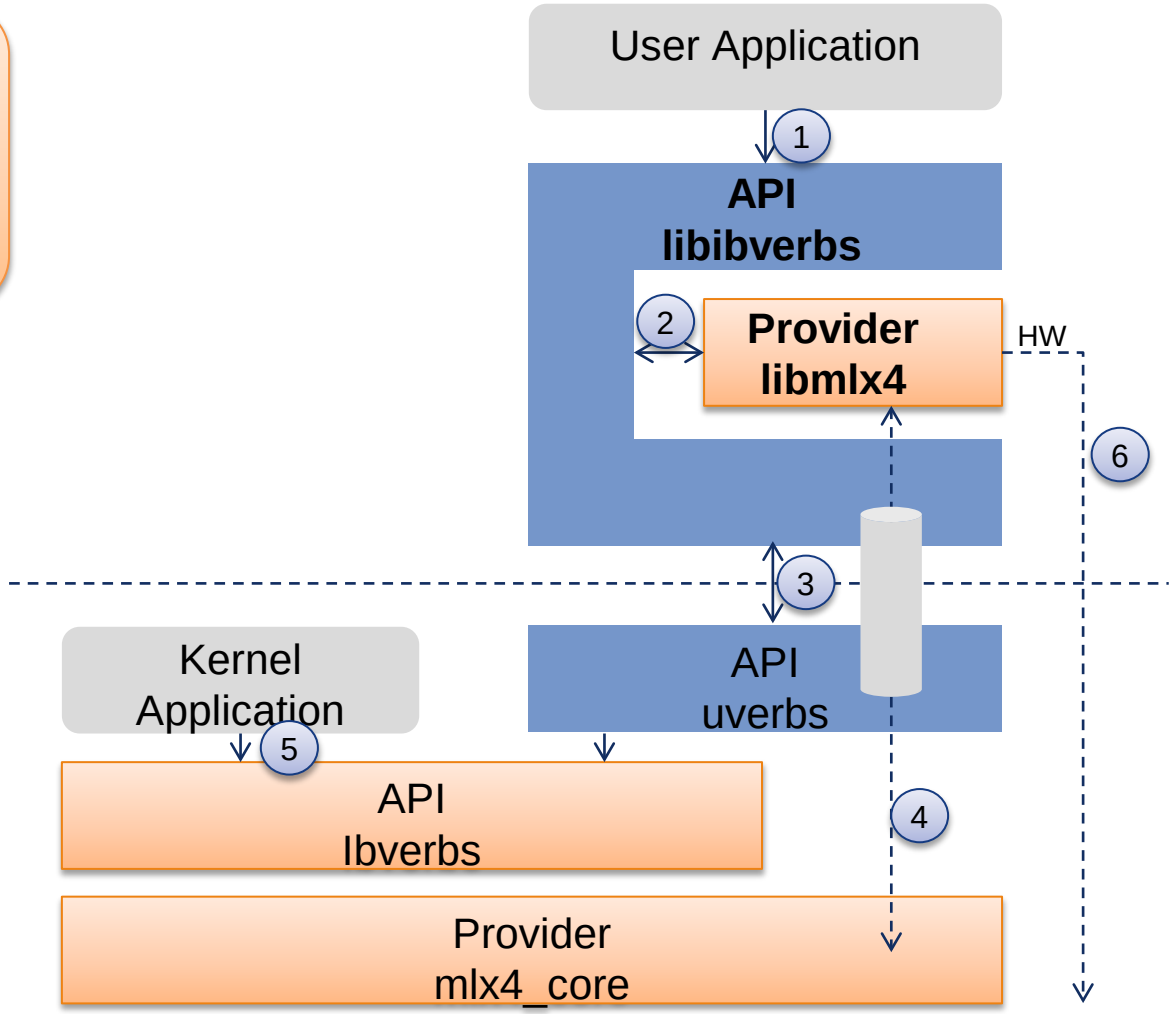
Code API App API	Header API Compile Time API	Lib API Run Time API	Supported
Old	Old	Old	**Y**
Old	Old	New	**Y**
Old	New	Old	**Y**
Old	New	New	**Y**
New	Old	Old	**N**
New	Old	New	**N**
New	New	Old	**Y**
New	New	New	**Y**

Backward and Forward Compatibility Factors

Factor	Non backward /forward compatible	Backward/forward compatible
API Functions	Modifying, removing, renaming and adding new function symbols	Function signatures left unmodified, no add or remove of new function symbols.
Function input arguments	Modifying compound or primitive argument type or size	Strict function arguments size, type and content - Unless input argument specifies new/old caller - Union can have new “meaning” but have strict size - Enumerations can include new values
Function return value, output parameter	Modifying function return value or output param type, size or meaning	Type, size and value should remain unmodified -Unless function output argument added new flags
Functions with dynamic allocations	API have asymmetric calls, ie. only have api_alloc() call and uses external free() call	Only if API support symmetric api_alloc() and api_free() calls

Stack Interfaces Map

1. User App and verbs API
2. API lib and provider lib
3. API lib and API kernel module
4. Provider user lib and kernel module
5. Kernel module and kernel verbs
6. Provider lib and direct HW access



Possible extension schemes

- .symver directive
  - Mangles ABI version through function symbol
  - Used today to support older Verbs v1.0
  - New apps will break with old lib
  - No function argument extensions
  - Symbol space will need to grow and grow
- OpenGL Extensions approach
  - Every extended call is wrapped with #ifdef
  - lib extended call support is tested by string match
  - Again, new apps will break with old lib
    - Extension supported is tested at compile time
  - Messy app code (#ifdef)
  - No function argument extensions

How

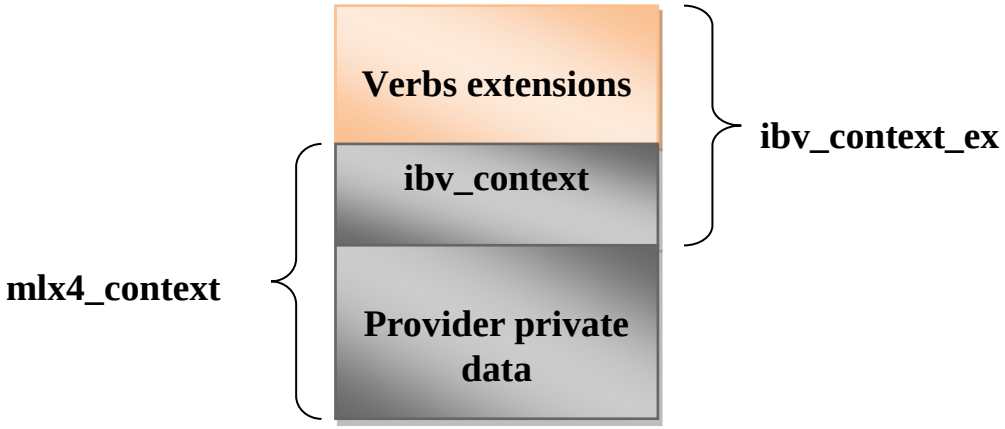
- New Verbs API calls
  - Are defined as inline in verbs.h
  - Will call library func pointer
  - Lib func pointer will call provider func pointer
  - Prior to extended func pointers call check for extensions
- For existing API calls
  - When new arguments are required
  - Add new call once with the name ibv_xxx_ex()
    - Using the scheme above
  - Old ibv_xxx() will use ibv_xxx_ex() for backward compatibility
- All is transparent to the Verbs App developer
  - Uses standard ibv_xxx() calls

How – Function Pointers

- Use function pointers - No new external symbols
  - For both library and provider Calls
- Extend ibv_context
  - Used as handle for almost all Verbs API calls
  - Used today to store provider calls only (in ibv_context_ops) in non-extendible way
    - Each addition completely changes the struct mem footprint
  - Re-use ibv_context.abi_compat
    - Used to support very old Kernel ABIs (versions <=2)
    - Instead, a marker of extended ibv_context - **ibv_context_ex**

Ibv_context_ex

- Extend struct ibv_context upwards in memory
  - Today provider uses ibv_context tail to store priv data
  - We will use ibv_context head to store extension data
- Define new struct ibv_context_ex
  - Wraps good old ibv_context
  - Extensions func pointers at the head and grows up
  - ibv_context at the bottom
  - ibv_context.abi_compat=-1 to indicate Verbs Extensions exist



Ibv_context_ex – Cont.

- Allocated by libibverbs
  - Through ibv_open_device()
  - Library to initialize lib func pointers
  - Provider to initialize provider func pointers through checking ibv_context_ex.sz

```
/* /libibverbs/include/infiniband/verbs.h */

struct ibv_context_ex {
    /*   ↑ "grows up" – new fields go here */
    int (*drv_new_func2) (…);
    int (*lib_new_func2) (…);
    int (*drv_new_func1) (…); /* new corresponding provider call of func1 */
    int (*lib_new_func1) (…); /* new library call func1 */
    uint64_t has_ex;          /* indicates extended calls, structs support */
    size_t sz;                /*  Set by library on struct allocation */
    struct ibv_context context; /* Must be last field */
};
```

How – Provider registration

- Providers registration libibverbs
  - Instead of ibv_register_driver() will use ibv_register_driver_ex()
  - New call will pass the provider's driver_init() extensions initialization callback and priv data sz
- driver_init() will be called on lib init
  - Provider will examine library extensions support
    - Through ibv_context_ex.sz
  - And will registers its func pointers extensions

Example

```
/* /libibverbs/include/infiniband/verbs.h */
/* Define function calls through simple wrapper inline function call */
static inline int ibv_new_func1(struct ibv_context *context, …)
{
     struct ibv_context_ex *ctx = ibv_get_ctx_ex(context);

     /* check if context is legacy ibv_context and not new ibv_context_ex */
     /* or if func1 is too new */
     if (unlikely((!ctx) ||
          (sizeof(*ctx) – offsetof(struct ibv_context_ex, lib_new_func1)
          > ctx->sz))
          return –ENOSYS; /* or call legacy API */

     return ctx->lib_new_func1(…);
}

/* /libibverbs/src/verbs.c */
static int __ibv_new_func1 (…) {
     struct ibv_context_ex *ctx = ibv_get_ctx_ex(context);

     ... /* library stuff */
     /* now call provider */

     if (unlikely(!ctx->drv_new_func1))
          return –ENOSYS; /* or call legacy API */
     else
          ctx->drv_new_func1(…);
}
```

How –Structures

- Structures usage
  - Input or output API call arguments
  - API call return value
- Build a scheme for extended structures
  - Support back and forward compatibility
  - Adding new fields will not break compatibility
- Structures to include *uint32 comp_mask* field
- *comp_mask*
  - Bit field indicating existence of the struct fields
  - MSB will indicate existence of comp_mask2
    - To support unlimited struct fields

How – Structures – Cont.

- For existing structures:
  - First see if structure is not already extendible
    - Has unions or flags
  - Define new struct ibv_xxx_ex
    - Which includes old fields
    - Followed by comp_mask field
    - Followed by new fields

```
struct ibv_xxx_ex {
    . . . /* Existing fields from ibv_aaa */
    uint32 comp_mask; /* Bit field indicating existing extended fields */
    struct ibv_aaa;  // New field
    struct ibv_bbb;  // New filed
};

enum ibv_xxx_ex_comp_mask {
    IBV_XXX_AAA = (1<<0), /* Marks ibv_xxx->ibv_aaa as valid */
    IBV_XXX_BBB = (1<<1), /* Marks ibv_xxx->ibv_bbb as valid */
    . . .
    IBV_XXX_COMP_MASK2 = (1<<32) /* Last mask bit indicates comp_mask2 */
}
```

How – Structures – Cont.

- Input arguments
  - comp_mask set by the caller (app) and read by the library
- Output arguments
  - comp_mask set by the library and read by the app
  - API call will pass structure pointer *plus* **size**
  - Library will use size argument to detect existing fields
- Arrays
  - Input or output args passed as an array will pass their size
  - Walk between elements (stride) will use the passed size

Example – Extended RC QP

```
/* /libibverbs/include/infiniband/verbs.h */

struct ibv_context_ex {
    /*  "grows up" - new fields go here */
    struct ibv_qp *  (*open_qp)(struct ibv_context *context,
                     struct ibv_qp_open_attr *attr);
    struct ibv_qp *  (*create_qp_ex)(struct ibv_context *context,
                     struct ibv_qp_init_attr_ex *qp_init_attr_ex);
    struct ibv_srq * (*create_srq_ex)(struct ibv_context *context,
                     struct ibv_srq_init_attr_ex *srq_init_attr_ex);
    struct ibv_xrcd *(*open_xrcd)(struct ibv_context *context,
                      struct ibv_xrcd_init_attr *xrcd_init_attr);
    int              (*close_xrcd)(struct ibv_xrcd *xrcd);
    uint64_t comp_mask;
    size_t   sz; /* Must be immediately before struct ibv_context */
    struct ibv_context context;/* Must be last field in the struct */
};
```

Example – Extended RC QP

```
struct ibv_xrcd {
    uint32_t comp_mask;
    struct ibv_context     *context;
};

struct ibv_xrcd_init_attr {
    uint32_t comp_mask;
    int   fd;
    int   oflags;
};

static inline struct ibv_xrcd * ibv_open_xrcd(struct ibv_context *context,
    struct ibv_xrcd_init_attr *xrcd_init_attr)
{
    struct ibv_context_ex *vctx = ibv_get_ctx_ex(context, open_xrcd);
    if (!vctx) {
        errno = ENOSYS;
        return NULL;
    }
    return vctx->open_xrcd(context, xrcd_init_attr);
}
static inline int ibv_close_xrcd(struct ibv_xrcd *xrcd);
```

Example – Extended RC QP

```
struct ibv_qp_init_attr_ex {
    ... /* Existing struct ibv_qp_init_attr fields */
    uint32_t     comp_mask;
    struct ibv_pd    *pd;
    struct ibv_xrcd  *xrcd;
};

static inline struct ibv_qp * ibv_create_qp_ex(struct ibv_context *context,
    struct ibv_qp_init_attr_ex *qp_init_attr_ex);

struct ibv_qp_open_attr {
    uint32_t          comp_mask;
    uint32_t          qp_num;
    struct ibv_xrcd  *xrcd;
    void              *qp_context;
    enum ibv_qp_type  qp_type;
};

static inline struct ibv_qp *ibv_open_qp(struct ibv_context *context, struct
    ibv_qp_open_attr *qp_open_attr)
```

Example – Extended RC QP

```
struct ibv_srq_init_attr_ex {
    . . . Existing ibv_srq_init_attr fields */
    uint32_t          comp_mask;
    enum ibv_srq_type srq_type;
    struct ibv_pd           *pd;
    struct ibv_xrcd         *xrcd;
    struct ibv_cq           *cq;
};

static inline struct ibv_srq *ibv_create_srq_ex(struct ibv_context *context,
    struct ibv_srq_init_attr_ex *srq_init_attr_ex);

struct ibv_send_wr {
    ...
    union {
        ...
        struct {
            uint64_t reserved[3];
            uint32_t reserved2;
            uint32_t remote_srqn; /* Used with IBV_QPT_XRC_SEND QP type */
        } xrc;
    } wr;
};
```

Example – Extended RC QP

```
/* Provider library registration mlx4.c */

static int mlx4_init_context(struct ibv_device_ex *v_device,
             struct ibv_context *ibv_ctx, int cmd_fd)
{
    struct ibv_context_ex *ibv_ctx_ex = ibv_get_ctx_ex(ibv_ctx);
    ...
    /*  Check ibv_ctx_ex->sz for verbs library support)
        In case supported, set the following */
    ibv_ctx_ex >comp_mask = IBV_CONTEXT_XRCD | IBV_CONTEXT_SRQ |
                    IBV_CONTEXT_QP;
    ibv_ctx_ex >close_xrcd = mlx4_close_xrcd;
    ibv_ctx_ex >open_xrcd = mlx4_open_xrcd;
    ibv_ctx_ex >create_srq_ex = mlx4_create_srq_ex;
    ibv_ctx_ex >get_srq_num = mlx4_get_srq_num;
    ibv_ctx_ex >create_qp_ex = mlx4_create_qp_ex;
    ibv_ctx_ex >open_qp = mlx4_open_qp;
    ...
}
```

Conclusions

- Application backward and forward compatibility
- Transparent to the developer
  - Verbs API has same look and feel
- Independent library and provider libs versions
- Simple guidelines for verbs, provider developers
- More and more features to come:
  - CQ time stamping, Flow Steering, …
- Next
  - Push verbs extensions infrastructure to upstream
  - Push Kernel uverbs extension scheme upstream
    - No need to define new uverbs CMD for new verbs or provider arguments
  - librdmacm API extensions
  - libibmad API extensions

Thanks

- Sean Hefty, Intel
- Jason Gunthorpe, Obsidian Research

Thank You

OPENFABRICS ALLIANCE

Example - Time Stamping

```
struct ibv_cq_init_attr {
     uint32 comp_mask;
     void *cq_context;
     struct ibv_comp_channel *channel;
     int comp_vector;
     int cq_create_flags; /* RAW or TOD time stamp */
};
struct ibv_cq *verbs_create_cq(struct ibv_context *context,
     struct ibv_cq_init_attr);

struct ibv_wc {
     ...
     union {
          int  wc_flags;
          int  comp_mask;
     }
     ...
     uint64_t            ts;
};
int ibv_poll_cq(struct ibv_cq *ibcq, int num_entries,
     struct ibv_wc *wc, int wc_size);

struct ibv_values {
     int       comp_mask;
     uint64_t  hwclock_ns;
};
int ibv_query_values(struct ibv_context *context, int q_values, struct *ibv_values
values);
```