OPENFABRICS ALLIANCE

Kernel OpenFabrics Interface

Initialization

Stan Smith Intel SSG/DPD

February, 2015

Steps

- Application Flow
  - Initialization*
  - Server connection setup
  - Client connection setup
  - Connection finalization
  - Data transfer
  - Shutdown

- Initialization
  - fi_getinfo() *info – query & select a fabric provider
  - fi_fabric() *fabric – create a fabric instance
  - fi_domain() *domain – select a fabric domain (which HCA via info)

kOFI Initialization

- #include <rdma/fabric.h>

- fi_getinfo() - Acquire a list of desirable/available fabric providers. List elements (struct fi_info) are filled-out per device/provider.
  - Release: fi_freeinfo(&fi);
  - Duplicate: fi2 = fi_dupinfo(&fi);

- Select appropriate fabric (traverse provider list).

- fi_fabric(fi, &fabric)
  Create a fabric instance based on fabric provider selection.

- fi_domain(fabric, fi, &domain) create a fabric access domain object.

fi_getinfo()

Acquire a list of available fabric providers based on struct fi_info hints.

```
int fi_getinfo(int version, const char *node, const int port,
               uint64_t flags, struct fi_info *hints, struct fi_info **info);
```

version - Interface version requested by application.
node - Optional, name or fabric address to resolve.
port - Optional, port number of address.
flags - Operation flags for the fi_getinfo call.
hints - *fi_info structure which specifies criteria for selecting fabrics.

info : (OUT) a linked list of fi_info structures containing fabric information.

fi_getinfo() changed

Acquire a list of available fabric providers based on struct fi_info hints.

int fi_getinfo(int version, struct fi_info *hints, struct fi_info **info);

version - Interface version requested by application.

Removed:

node - Optional, name or fabric address to resolve.

port - Optional, port number of address.

flags - Operation flags for the fi_getinfo call.

In favor of hints.src_addr -> struct sockaddr for (RDS & IB providers)

hints - *fi_info structure which specifies criteria for selecting fabrics.

info : (OUT) a linked list of fi_info structures containing fabric information.

fi_getinfo() cont.

```
struct fi_info {
        struct fi_info      *next;
        uint64_t            caps;           // fabric interface capabilities
        uint64_t            mode;           // Operational modes supported by the IF
        enum fi_ep_type ep_type;            // type of fabric IF communication desired
        uint32_t            addr_format;    // format of addresses referenced by the  fabric
        size_t              src_addrlen;    // length of the source address
        size_t              dest_addrlen;   // length of the destination address
        void                *src_addr;      // source address
        void                *dest_addr;     // destination address
        fi_connreq_t        connreq;        //  a specific connection request
        struct fi_tx_attr   *tx_attr;
        struct fi_rx_attr   *rx_attr;
        struct fi_ep_attr   *ep_attr;
        struct fi_domain_attr *domain_attr;     // optional: .name = “ib0”
        struct fi_fabric_attr *fabric_attr;
};
```

Tx attributes

Optional Transmit context attributes may be specified and returned as part of fi_getinfo. When provided as hints, requested values of struct fi_tx_ctx_attr should be set. On output, the actual transmit context attributes that can be provided will be returned.

```
struct fi_tx_attr {
        uint64_t    caps;           # see CAPABILITIES section in fi_getinfo(3) for details
                                    # e.g. FI_MSG, FI_RMA, FI_ATOMIC
        uint64_t    mode;           # see MODE section in fi_getinfo(3) for details; FI_LOCAL_MR
        uint64_t    op_flags;       # Tx operation types: (man fi_endpoint.3)
        uint64_t    msg_order;      # order in which transport layer headers (as viewed by the
                                    # application) are processed.
                                    FI_ORDER_NONE, FI_ORDER_STRICT
        uint64_t    comp_order;     # order in which completed requests are written into the
                                    # completion queue.
        size_t      inject_size;    # max Tx buffer size which is reusable prior to associated CQ event.
        size_t      size;           # bytes in local Tx context queue; work-queue depth.
        size_t      iov_limit;      # max QP scatter-gather list depth (num elements).
        size_t      rma_iov_limit;  # max RDMA scatter-gather list depth.
};
```

Rx attributes

rx_attr - Optional receive context attributes may be specified and returned as part of fi_getinfo. When provided as hints, requested values of struct fi_rx_ctx_attr should be set. On output, the actual receive context attributes that can be provided will be returned. Output values will be greater than or or equal to the requested input values.

```
struct fi_rx_attr {
        uint64_t    caps;               # see CAPABILITIES section in fi_getinfo(3) for details
                                        # e.g. FI_MSG, FI_RMA, FI_ATOMIC
        uint64_t    mode;               # see MODE section in fi_getinfo(3) for details; FI_LOCAL_MR
        uint64_t    op_flags;           # permitted operations
        uint64_t    msg_order;          # order in which transport layer headers are processed.
        uint64_t    comp_order;         # order in which completed requests are written into the
                                        # completion queue.
        size_t      total_buffered_recv; # total size of provider managed Rx buffer space.
                                        # (see FI_BUFFERED_RECV context flag)
        size_t      size;               # bytes in local Rx context queue; work-queue depth.
        size_t      iov_limit;          # max scatter-gather list depth.
};
```

EndPoint attributes

ep_attr - Optional endpoint attributes may be specified and returned as part of fi_getinfo. When provided as hints, requested values of struct fi_ep_attr should be set. On output, the actual endpoint attributes that can be provided will be returned. Output values will be greater than or equal to requested input values.

```
struct fi_ep_attr {
        uint32_t        protocol;           # end-2-end protocol: FI_PROTO_RDMA_CM_IB_RC
        uint32_t        protocol_version;   # connection protocol version
        size_t          max_msg_size;       # max number of bytes per data transfer operation.
        size_t          inject_size;        # max TX buffer size that can be reused prior to CQ event.
        size_t          total_buffered_recv; # Total # of Rx bytes utilized for FI_BUFFERED_RECV
        size_t          msg_prefix_size;    # size of message prefix size(0), FI_MSG_PREFIX
        size_t          max_order_raw_size; # RDMA: Read-after-Write byte size
        size_t          max_order_war_size; # RDMA: Write-after-Read byte size
        size_t          max_order_waw_size; # RDMA: Write-after-Write byte size
        uint64_t        mem_tag_format;     # bit array: number of tag bits supported by the provider
        uint64_t        msg_order;          #  order in which transport layer headers are processed.
        uint64_t        comp_order;         # order which completed requests are written to the CQ.
        size_t          tx_ctx_cnt;         # Tx work-queue-depth
        size_t          rx_ctx_cnt;         # Rx work-queue-depth
};
```

Domain attributes

domain_attr - Optionally supplied domain attributes may be specified and returned as part of fi_getinfo. When provided as hints, requested values of struct fi_domain_attr should be set. On output, the actual domain attribute that can be provided will be returned. Output values will be >= to requested input values.

```
struct fi_domain_attr {
        struct fid_domain       *domain;  # optional existing domain instance
        char                    *name;    # ib0, /sys/class/infiniband
        enum fi_threading       threading;
                FI_THREAD_UNSPEC, FI_THREAD_SAFE, FI_THREAD_FID,
                FI_THREAD_DOMAIN, FI_THREAD_COMPLETION, FI_THREAD_ENDPOINT
        enum fi_progress        control_progress;
        enum fi_progress        data_progress;
                FI_PROGRESS_UNSPEC, FI_PROGRESS_AUTO (provider determined),
                FI_PROGRESS_MANUAL (provider requires the use of an application thread to complete
                                    an  asynchronous request (socket dev); eq/wait/cq/counter).
<…>
};
```

Domain attributes II

```
struct fi_domain_attr {
        <…>
        enum fi_resource_mgmt  resource_mgmt;
                        # provider support for protecting local and/or remote resource overruns; IB RNR
                        FI_RM_UNSPEC, FI_RM_DISABLED, FI_RM_ENABLED
        size_t          mr_key_size;        # range in bytes a key may span
        size_t          cq_data_size;       # remote CQ data size, Tx supplied RCQ data
        size_t          cq_cnt;             # max num of CQs per domain
        size_t          ep_cnt;             # max num of EndPoints per domain
        size_t          tx_ctx_cnt;         # optimal num of Tx QPs
        size_t          rx_ctx_cnt;         # optimal num of Rx QPs
        size_t          max_ep_tx_ctx;      # max num of Tx contexts per EP
        size_t          max_ep_rx_ctx;      # max num of Rx contexts per EP
};
```

Fabric attributes

fabric_attr - Optional fabric attributes may be specified and returned as part of fi_getinfo. When provided as hints, requested values of struct fi_fabric_attr should be set. On output, the actual fabric attributes that can be provided will be returned.

```
struct fi_fabric_attr {
        struct fid_fabric       *fabric;        # optional already opened fabric instance,
        char                    *name;          # infiniband, /sys/class/
        char                    *prov_name;     # ibverbs
        uint32_t                prov_version;
};
```

RDS Example

```
struct fi_info          *providers, hints = { 0 };
struct fi_fabric_attr   fabric_attr = { 0 };
struct fi_domain_attr   domain_hints = { 0 };
struct fi_ep_attr       ep_hints = { 0 };

ep_hints.protocol    = FI_PROTO_UNSPEC;
fabric_attr.name     = "IP";
fabric_attr.prov_name = "RDS";
      // domain_hints.name = "ib0";  // provider will set local address in hints.src_addr
hints.domain_attr    = &domain_hints;
hints.fabric_attr    = &fabric_attr;
hints.ep_attr        = &ep_hints;
hints.ep_type        = FI_EP_RDM;        /* Reliable Datagram */
hints.caps           = FI_MSG |  FI_SOURCE | FI_CANCEL | FI_SEND | FI_RECV;
hints.addr_format    = FI_SOCKADDR_IN;
hints.src_addr       = (struct sockaddrs_in*) &local_IF_addr;
rc = fi_getinfo( FT_FIVERSION, &hints, &providers );
```

Infiniband RC Example

```
struct fi_info          *provider, hints = { 0 };
struct fi_fabric_attr   fabric_attr = { 0 };
struct fi_domain_attr   domain_hints = { 0 };
struct fi_ep_attr       ep_hints = { 0 };

ep_hints.protocol   = FI_PROTO_RDMA_CM_IB_RC;
hints.ep_type       = FI_EP_MSG;
hints.caps          = FI_MSG | FI_RMA | FI_CANCEL;
hints.addr_format   = FI_SOCKADDR_IN;
hints.src_addr      = &local_IF_addr;
hints.src_addrlen   = sizeof(local_IF_addr);

local_IF_addr.sin_family        = AF_INET;
local_IF_addr.sin_port          = htons(SVR_PORT);
local_IF_addr.sin_addr.s_addr = htonl(SVR_ADDR);      // or in4_pton()

ret = fi_getinfo(FT_FIVERSION, &hints, &provider);
```

Concerns fi_info()

- Node name resolution for each provider?
- Failure to expose port configuration, sans doman.name?
  fi_info per port? Sockaddr_ib.
- Complexity – trying to be everything to everyone; how to reduce?
  A struct fi_info* for each:
    - EndPoint type: IB: RC, UD
    - Fi_info for each port, although no explicit port identification?

fi_fabric()

Open the fabric domain provider which the application selected.

A fabric domain represents a collection of hardware and software resources that access a single physical or virtual network.

```
struct fid_fabric       *fi;

int fi_fabric( fi_fabric_attr *attr, struct fid_fabric *fabric, void *context )
```

Arguments:

attr: Attributes of fabric to open.

fabric (out) Fabric provider pointer

context: fabric level event context.

fi_domain()

Open a fabric access domain. A fabric access domain typically refers to a physical or virtual NIC or hardware port; however, a domain may span across multiple hardware components for fail-over or data striping purposes.

```
struct fid_domain     *domain;

int fi_domain(struct fid_fabric *fabric, struct fi_info *info,
                        struct fid_domain **domain, void *context);
```

Arguments:

- fabric – previously opened fabric pointer.
- info – provider info structure used in creating the 'fabric' pointer.
- domain (out) fabric provider domain.
- context – domain event context.

Example Application struct

```
typedef struct {
        struct fi_context       context;
        struct fi_info          *prov;
        struct fid_fabric       *fabric;        // set during initialization steps
        struct fid_domain       *domain;        // set during initialization steps
        struct fid_domain       *sdomain;       // server domain.
        struct fid_ep           *ep;
        struct fid_pep          *pep;
        struct fid_eq           *eq;
        struct fid_cq           *scq;
        struct fid_cq           *rcq;
        struct fid_mr           *mr;
        char                    *buf;
} application_context_t;

application_context_t       ctx = { 0 };
```

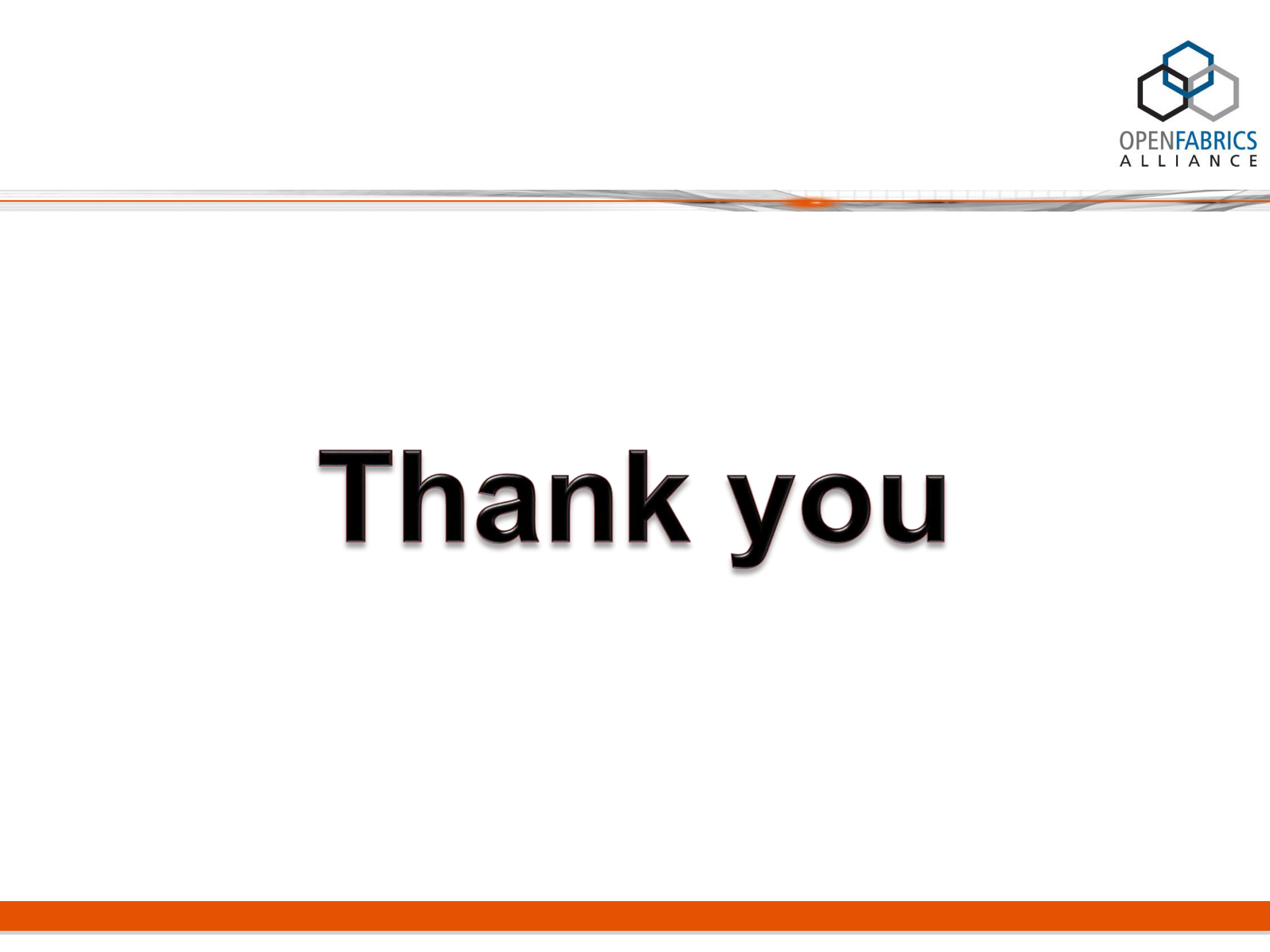
Examples

```
struct fi_info          *fi =  provider fi_info record selected from fi_getinfo() provider list
int                     rc;

rc = fi_fabric( fi->fabric_attr, &ctx.fabric, NULL );
if ( rc ) {
        printk( KERN_ERROR “%s() ERR: fi_fabric() rc %d\n”,__func__,rc);
        return rc;
}

rc = fi_domain( fabric, fi, &ctx.domain, NULL );

// Ready to create End points: passive and active to connect.
```


OPENFABRICS
ALLIANCE
Thank you