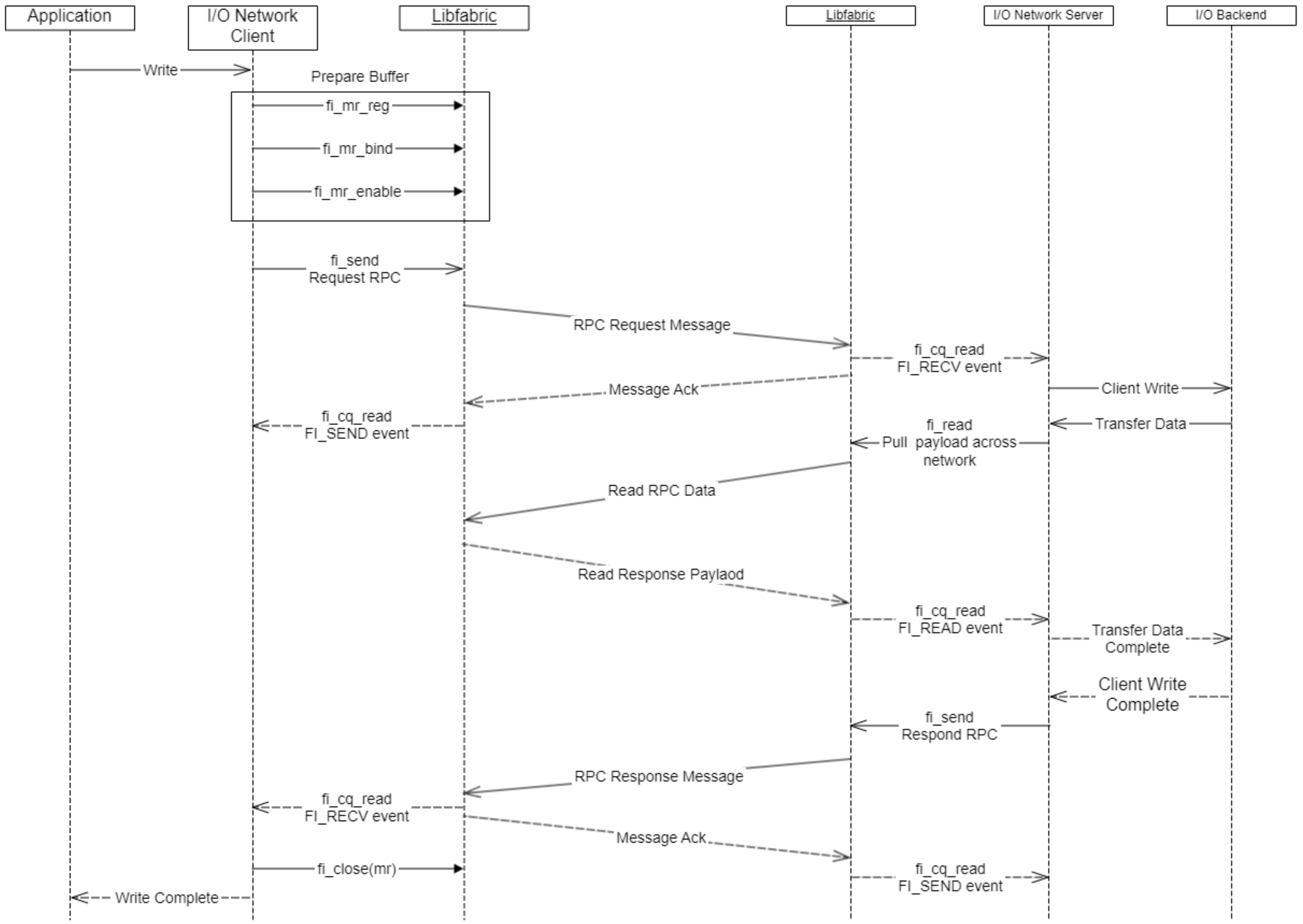
Hewlett Packard
Enterprise

Libfabric Tagged RMA

Ian Ziemba, Software Engineer

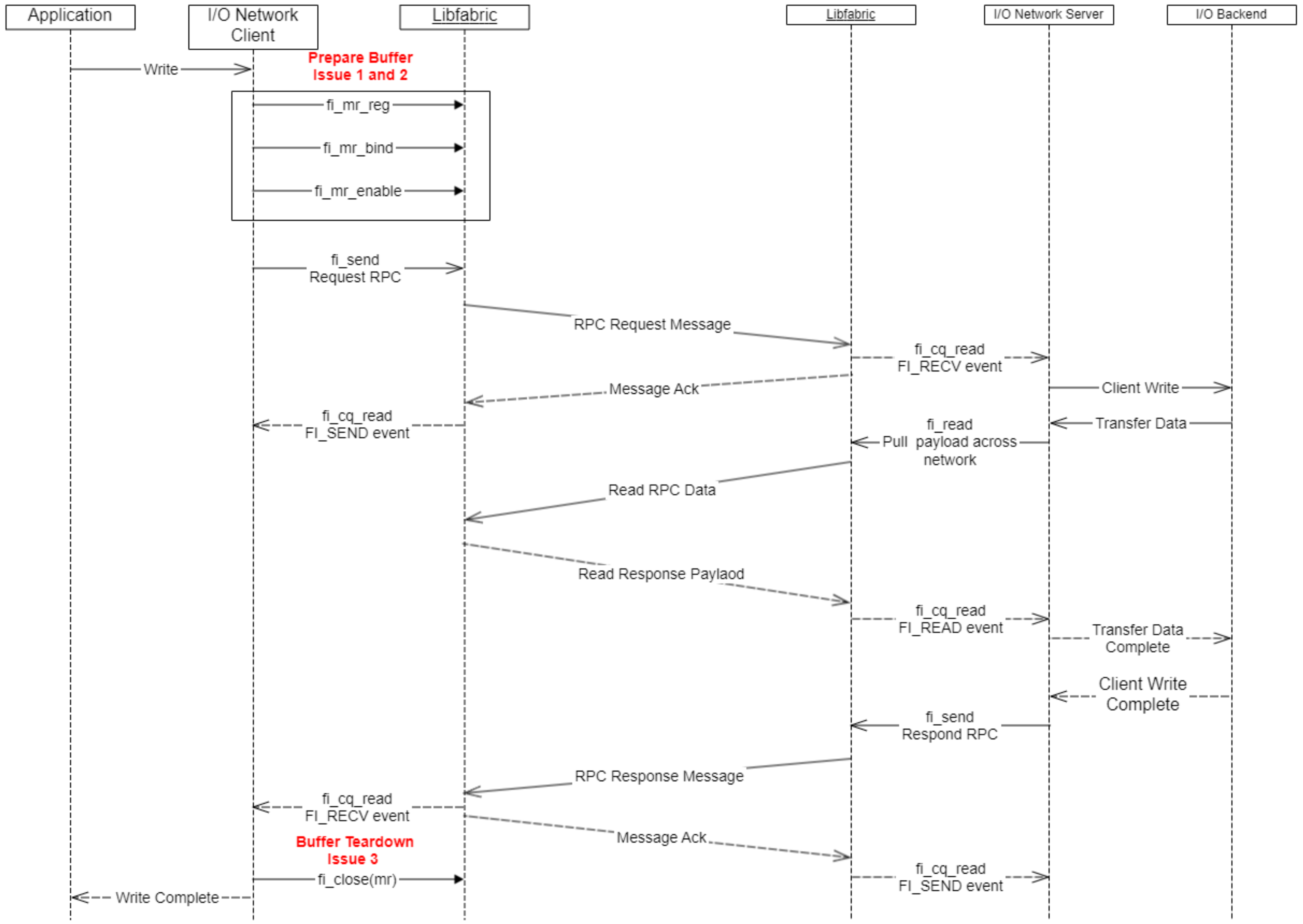
Storage Libfabric Use Case: Overview

- RCP style of communication
- Client operations
  - Prepare payload buffer
    - Libfabric operation: memory registration
  - Send RPC request
    - Libfabric operation: messaging
  - Wait for RPC response
    - Libfabric operation: messaging
- Server operations
  - Wait for RPC request
    - Libfabric operation: messaging
  - Transfer client payload
    - Libfabric operation: RMA
  - Execute RPC
  - Send RPC response
    - Libfabric operation: messaging



Storage Libfabric Use Case: Issues/Areas of Optimizations

- Issue 1: Expensive per RPC memory registration
  - Need MR caching of remote MRs to avoid memory registration cost
    - May force libfabric users to run with remote MRs cached
  - Can result in a client leaving memory exposed in multi-client, single server environment
    - Different authorization keys between clients can help
    - Requires server to operate on multiple authorization keys
- Issue 2: MR cache and RDM endpoints
  - RDM endpoints do not require connection establishment
  - Cannot restricted MR to specific RDM endpoint
- Issue 3: Synchronous teardown of MR resources
  - Can be expensive (provider dependent)
  - Need MR cache to avoid memory deregistration overhead



Storage Libfabric Use Case: Proposal

- Requirements
  - Interface which enables read operations to use-once provider resource(s)
  - For RDM endpoints, enable matching on source address for read operation
  - For non-FI_MR_LOCAL providers, do not require explicit memory registration
- Proposal: Extend tagged AMO API to RMA (tagged RMA)
  - FI_TAGGED AMO: Specifies that the target of the atomic operation is a tagged receive buffer instead of an RMA buffer. When a tagged buffer is the target memory region, the addr parameter is used as a 0-based byte offset into the tagged buffer, with the key parameter specifying the tag.
  - Initiator adds in the FI_TAGGED to AMO operation
  - Target posts normal tagged buffers

Storage Libfabric Use Case: Tagged RMA Overview

- Definition: FI_TAGGED specifies that the target of the RMA operation is a tagged receive buffer instead of an RMA buffer. When a tagged buffer is the target memory region, the addr parameter is used as a 0-based byte offset into the tagged buffer, with the key parameter specifying the tag.
  - New capability FI_TAGGED_RMA
  - FI_TAGGED is passed into the fi_writemsg/readmsg
  - Open Question: Should FI_RMA be passed into fi_trecvmsg?
    - Does this diverge from FI_TAGGED AMO API?
  - Open Question: Should tagged RMA write be supports?
    - Seems to match to normal tagged send
- Benefits compared to "use-once" MRs + traditional RMA
  - Provider automatically tears down target resources
    - Potentially avoids some overhead in explicit fi_close(mr) calls
  - Improved security
    - Tagged receive buffers are only network accessible for a single operation
      - MR cache can leave remote MRs exposed to the network indefinitely
    - Enables matching on source address (initiator)
      - With RDM endpoints, remote MRs are exposed to all initiators
      - MR cache leaves remote MRs exposed to all initiators

Storage Libfabric Use Case: Tagged RMA Event Generation

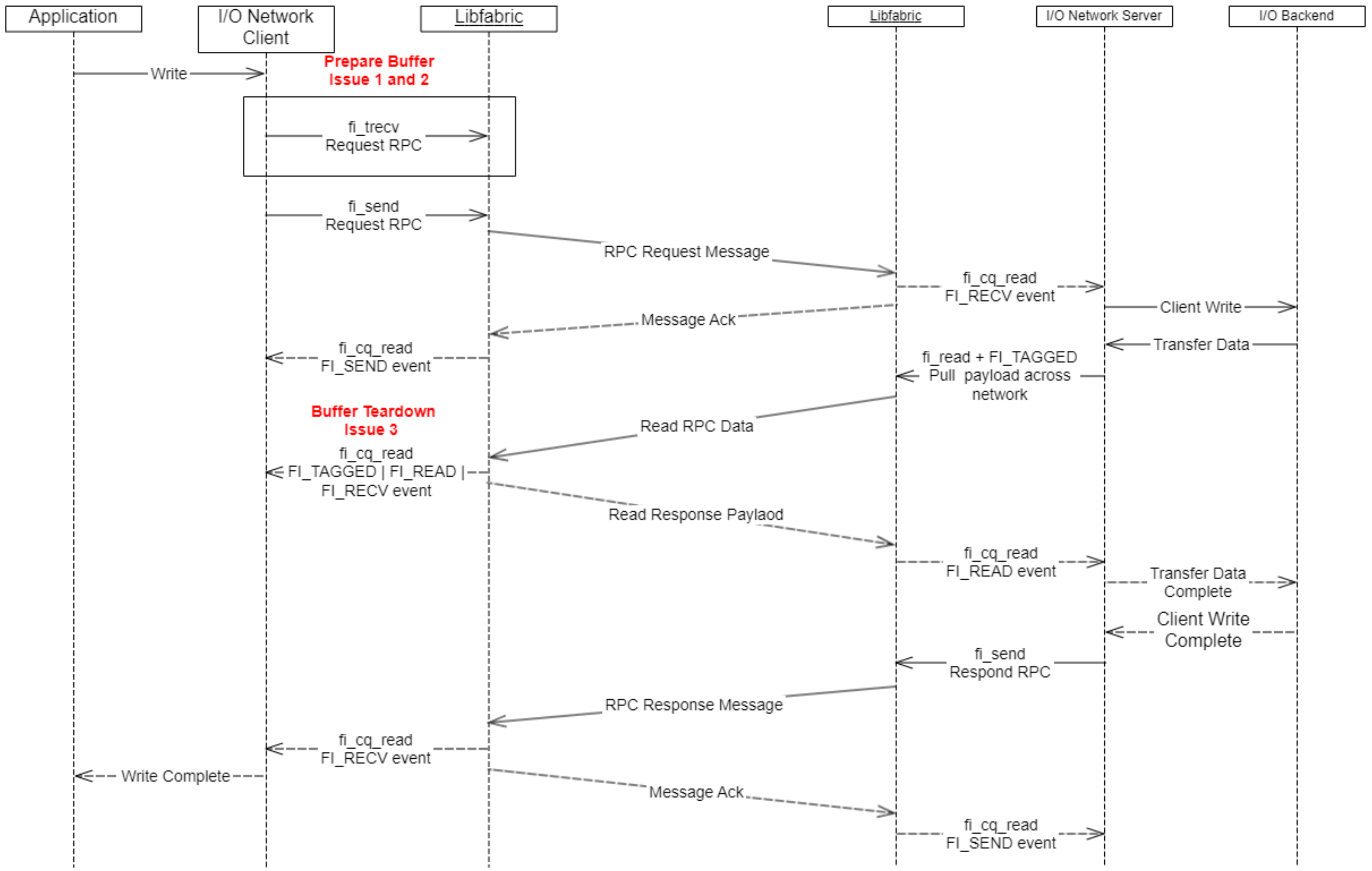
- Tagged messages have initiator and target events
- Open Question: Since tagged RMA operations target a tagged receive buffer, should target event generation be support?
  - Tagged RMA target events could enable higher-level application progression without additional operation round trip
    - Example: FI_TAGGED | FI_READ | FI_RECV (tagged read target event) event with user provided context
- Open Question: When would the tagged RMA read target event get generated? Is it when the tagged buffer will not be reused? Something else?
  - Are different levels of completion needed for tagged RMA read target events?
- Open Question: Should remote CQ data be supported for tagged RMA read?
- Open Question: Can counters be supported with tagged RMA read?
- Note: These questions could extend to normal RMA API as well.

Storage Libfabric Use Case: Unexpected Tagged RMA

- Using tagged buffers for RMA opens the door for unexpected tagged RMA
  - fi_trecv asynchronously posts buffers to the provider
- Open Question: Are new message ordering flags needed for tagged RMA (i.e. FI_ORDER_TAGGED_RMA_{RAR|WAR|RAW|WAW})
  - Enables provider to treat messaging, RMA, and tagged RMA separately
  - Enables libfabric users to select desired behavior
- Resource management
  - Enabled: Provider/hardware internally retry within some time window (i.e. RNR timeout)
  - Disabled: Provider/hardware returns undefined error

Storage Libfabric Use Case: Updated Issues

- Issue 1: Expensive per RPC memory registration
  - Resolve by async tagged buffer posting
  - Cache can still be implemented for local MRS
- Issue 2: MR cache and RDM endpoints
  - Tagged buffers can be limited to a single source address
- Issue 3: Synchronous teardown of MR resources
  - Tagged buffers are use-once



Storage Libfabric Use Case: Loopback Tagged RMA Example

```c
struct fi_info *hints = fi_allocinfo();

/* Client defines whatever hints they want. Assume these are set. Only
 * FI_TAGGED_RMA is only called out in this example.
 */
hints->caps |= FI_TAGGED_RMA;

/* Assume endpoint is allocated with this capable and a loopback address
 * to this endpoint has been allocated.
 */
struct fid_ep *ep;
fi_addr_t loopback;

/* Assume completion queue is allocated and bound to the endpoint. */
struct fid_cq *cq;

/* Buffers used for RMA with FI_TAGGED. */
char source[4096];
char target [4096];

/* Post the tagged buffer to be the target of the RMA with FI_TAGGED. */
uint64_t tag = 0x12345;
fi_trecv(ep, target, 4096, NULL, loopback, tag, 0, NULL);
```

```c
/* Issue RMA write with FI_TAGGED to the posted tagged buffer. */
struct iovec iov = {
    .iov_base = source,
    .iov_len = 4096,
};
struct fi_rma_iov rma_iov = {
    .len = 4096,
    .key = tag,
};
struct fi_msg_rma rma_msg = {
    .msg_iov = &iov,
    .iov_count = 1,
    .addr = loopback,
    .rma_iov = &rma_iov,
    .rma_iov_count = 1,
};
fi_readmsg(ep, &rma_msg, FI_TAGGED | FI_COMPLETION);

/* Poll for source and target completion events. */
unsigned int event_count = 0;
struct fi_cq_tagged_entry event;
while (event_count < 2) {
    if (fi_cq_read(cq, &event, 1) == 1) {
        /* Got initiator event. */
        if (event.flags == (FI_TAGGED | FI_READ | FI_SEND))
            event_count++;
        /* Got target event. */
        else if (event.flags == (FI_TAGGED | FI_READ | FI_RECV))
            event_count++;
        else
            abort();
    }
}
```

Thank you

ian.ziemba@hpe.com