Overview

1. Working with Subgroups
2. Derecho memory model
3. Send and Delivery
4. More on SubgroupInfo
To run operations on subgroups, we need handles to the subgroups. They are provided by Group’s function `get_subgroup`:

```cpp
1 template <typename SubgroupType>
2 auto& get_subgroup(uint32_t subgroup_index = 0);
```

The function is templated on the type `SubgroupType`. Its argument `subgroup_index` selects one among possibly many subgroups of that type.

You can only get a handle if you replicate that subgroup’s object. In other words, you must be a member of that subgroup. The function throws an exception, if that is not the case.
Derecho maintains a pool of buffers for every subgroup.

This is registered in advance with RDMA (for network operations).

The application must query Derecho for a memory buffer, generate the message in it and then call send. This needs to happen through the subgroup handles.

A memory buffer for a subgroup is busy from when it is provided to the application to when the message is delivered by all members of the subgroup.

Thus if a subgroup has no free buffers available (because all buffers are being used for past messages), it must wait!
These functions must be called one after another. If you want to send two messages: Ask for a buffer, send first message, then ask for another buffer and send the second message. Do not ask for two buffers at once.

payload_size is the size of the message you’re sending. It must not exceed the maximum message size, provided as derecho parameters.
Example

```c
// get a handle to the subgroup
RawSubgroup& subgroupHandle = group->get_subgroup<
    RawObject>();

// ask for a buffer
char* buf = subgroupHandle.get_sendbuffer_ptr(6);
// busy wait until you get a valid buffer
while(!buf) {
    buf = subgroupHandle.get_sendbuffer_ptr(6);
}

// generate the message
buf[0] = H;
...

// call send
subgroupHandle.send();
```
Delivery callback

- After messages are sent in a subgroup, they are delivered at all members of the subgroup. The order of delivery of the messages across all members is the same.

- The application is notified of delivery through a delivery callback by Derecho. The group constructor takes the callback as one of its arguments. Delivery callback is a way for the application to take concrete actions on message delivery, knowing that the message will be delivered at other members in the same order.

- See the callback in the sample_application.cpp file
Example

- Suppose we have two subgroup types: Foo and Bar.
- Foo has only one subgroup, Bar has two.
- Foo’s only subgroup has two shards, one contains all even-ranked members of the top-level group i.e., 0, 2, 4, …. The other contains all members with odd-ranks i.e. 1, 3, ….
- Bar’s first subgroup has a single shard. It contains the first three members of the group.
- Bar’s second subgroup has three shards. The first contains members with rank (in the top-level membership) 0, 1, 4. The second contains members with rank 2, 3, 5. The third contains members with rank 7, 8.
Subgroup features

- What we have defined is a membership allocation function for the subgroups as a function of the top-level group membership.
- Note that some subgroup memberships aren’t defined. Bar’s first subgroup’s only shard needs the group to have at least 3 members. Similarly, Bar’s second subgroup’s third shard needs the group to have at least 9 members (otherwise node with rank 8 will not exist).
- You can throw a derecho::subgroup_provisioning_exception() if you cannot provision a subgroup (likely because you don’t have enough members).
- Shards of different subgroups can overlap in the members list, but shards of the same subgroup must be disjoint.