**OFI WG Bi-Weekly telecom – 10/20/2015**

**Agenda:**

* Roll call, agenda bashing
* Travis testing – how we deal with external libraries
* Request to do function tracing
* Improving support for synchronous operations
* Update on work being done on utility provider
* Question about mode bits in fi\_getinfo

**Travis Testing – how do we deal with external libraries**

- Is there a policy for which version of e.g. a particular provider, should be built and/or installed?

- Proposal – go with latest GIT tree release. If that turns out to be a problem, we can explore changing the scripts to pick a particular release.

**Request to do Function Tracing**

- Had a request to be able to trace all function entry points into the library, and possibly dump all entry parameters. This would be entry points into the library, not all functions within the library.

- Possibly construct a proxy provider controlled by the usual logging environment variables that would insert itself between the consumer and the actual provider. The idea is that the proxy provider would hook all the calls to the actual provider. This would let us e.g. time stamp calls, or whatever.

- Possibly use LTTNG – Linux Tracing Toolkit Next Generation (user space version of it). Sort like a little library that inserts trace points for you.

- Does LTTNG require mods to the application? Ans – no, it shouldn’t, but we might have to modify our dispatch functions slightly.

- In general, the idea of injecting a tracing ability seems like a very useful idea.

- No conclusions today. Bring this back for more discussion in a couple of weeks, after looking at LTTNG to see what it can provide. But there’s no easy way to display a bunch of parameters for a function, other than just typing it in.

**Update on Work Being Done on Utility Provider**

- The ‘Utility Provider’ would be UDP-based since UDP provides the minimum functionality. See the branch in Sean’s github tree.

- The hope, at least in the beginning, is that the utility provider would supply some common features from which provider writers could draw.

- This would add things that aren’t well supported today, such as multicast.

- This is based on the idea that the framework would provide features that could be used by any particular provider that doesn’t provide that particular functionality.

- Today, a UDP provider is being created, along with the utility provider. But today, the utility provider isn’t a true standalone provider. Instead the utility provider is a repository for a series of routines that the UDP provider is using. Not clear if we will end up with a ‘utility provider’ on top of an existing provider, or it is simply a library of routines that another provider can pull into its own code base to make it easier for a provider writer. It may also make it easier on the application writer, which doesn’t have to figure out if a particular function is provided, he will just know that the function is always there.

- Not yet clear if this is actually a new provider, or is an internal library.

**Mode bits in fi\_getinfo**

- man pages are a bit unclear here. If the caller calls fi\_getinfo and sets the mode bits to zero, is the provider supposed to return the optimal set of modes, or all the modes that it supports, or should be assuming that the caller supports no modes?

- if the application passes in no mode bits, the assumption should be that the caller supports no modes.

**Synchronous Operations**

- Currently, the interfaces are all defined as asynchronous. But when you have a provider that does any of the work in s/w, e.g. tag matching or atomic operations in s/w, there is overhead that can occur.

- Possible solution would be to have e.g. the send function complete synchronously, or have CQs tied to callbacks…

- Most general solution would be to add a callback to the completion queue.   
- Problem – callbacks have their own sets of issues, e.g. what thread context are you calling in? What thread is invoking the callback is the problem. Not clear if a callback mechanism makes sense if you’re doing a thread context switch, since you’d still end up building a structure of some sort and putting it on a queue somewhere.

- But if you have a h/w-based tag matching mechanism, how do you have the h/w invoke a call back efficiently without losing the gains of the hardware?

- Using the existing progress model (auto vs manual), you would only use the callback if the progress model was set to manual.

- MPI hasn’t done this because nobody can agree on what you can do in the callback.

- To be continued at the next meeting.

**Future Agenda Topics:**

**Webex link:** <https://cisco.webex.com/ciscosales/j.php?MTID=m9389b0513c9ae643d57e2381e254dcf5>  
Webex password: ofi

**OFIWG Download Site:** [www.openfabrics.org/downloads/OFIWG](http://www.openfabrics.org/downloads/OFIWG)

**Github:** <https://github.com/ofiwg/libfabric>

**OFI Software Download Site:** [www.openfabrics.org/downloads/OFI](http://www.openfabrics.org/downloads/OFIWG)

**Link to WebEx Recording** - [**Play recording**](https://cisco.webex.com/ciscosales/lsr.php?RCID=39f67f25aeaf4907b5af0c0b1ede1a30)

**Next regular telecon**

Next meeting: Tuesday, 11/03/15

9am-10am Pacific daylight time