**OFI WG Bi-Weekly telecom – 07/28/2015**

**Agenda:**

* Roll call, agenda bashing
* fabtests for different versions of libfabric
* completion flags #1142
* Rel 1.1
* other topics

**Release 1.1 update**

- Resolving how to handle a long double is the blocking issue at this point. The industry has more or less decided that long doubles are not especially useful, but we use a long double in atomics. The question is how to make this useful?

- Proposal – for floating point, define the operations as float 32, float 64, float 96, float 128.

- Current patch omits 96 bit value, but with the 128 bit version it compiles.

- MPI more or less punts by using long double and hoping that it gets compiled with the same library. Problems may exist if MPI and MPI library are compiled with different compiler options. So to some extent, MPI is exposed to this issue already, independent of OFI.

- Should we do the same? The conventional approach these days seems to be to punt. The downside is that in the meantime someone may decide to start trying to use it.

- At the very least, the application needs to know the size – i.e. a long double is 16 bytes. But don’t reuse the name ‘long double’, (which is why Sean invented the names e.g. float96.)

- Consensus – defer the solution until 1.2.

- Rupert – EWG is planning a point release of 3.18.1. Sean to send a note to Vlad to pull rel1.1 rc2.

- If we defer the atomic stuff, release 1.1 will probably be out in the next few days.

- A new test has been added to fabtest, ‘setname’, but this may cause problems for other providers. Therefore there is a question as to whether or not to include this new test in fabtest for release 1.1

**Reviewing results from last meeting - Fabtests for different version of libfabric - Sean**

Conclusion: fabtest and libfabric will be kept in sync. This means that any given version of fabtests may not work with earlier versions of libfabric; i.e. to test libfabric you need to find the corresponding version of fabtest.

**Completion Flags – Sean GitHub issue #1142**

When a data operation completes, we can report information about the type of request that it was for. E.g. was it a transmit or receive operation? This can be extended, for example, was the transmit a send or RMA write? Did it have remote CQ data (immediate data)? etc. up to capturing the full amount of information about the request. The question is what amount of information will apps actually find useful?

Currently completion flags are defined which can map well to something like an opcode. The more this can be relaxed, the more optimization possibilities there are.

Basic question: How much information should be presented to the consumer via completions? Providing added information may cause branches/extra time to gather and present.

Should there be some mechanism that reflects to the software (consumer) that providing additional information is not free?

Consumers (e.g. UPC compiler writers) don’t want to have to include multiple ifdefs, one per provider. One the other hand, they do want some sort of mechanism to reflect to the consumer that this information is not free by forcing it to explicitly request certain completion fields.

Having raised awareness of the issue, probably appropriate to take this back to GitHub.

Open question – are all providers required to support all completion types?

If not, how do you allow an application to learn about unsupported formats at fi\_get\_info time, instead of at create\_completion time?

From a consumer perspective, it’s obviously much easier if it can rely on all providers supporting all completion types and the flags. But it’s not clear that all providers can (easily) do so.

AR to all provider vendors: Review the list of flags described in the man pages (fi\_cq3, fi\_msg).

**Porting libfabric on other platforms (Windows?) - Sean**

- libfabric would be layered over MSFT windows direct.

- But what would be the point? It would be useful for an application that wants to write to single API to be run over either Windows or Linux. Rather than write to windows direct, the application would be written to libfabric.

- Are there examples of applications that would fall in this category? Not exactly.

- Today, MSFT relies mainly on DAPL.

- The question becomes, do we want libfabric to become the next DAPL? (No, not really)

- Is there anyone willing to do the work? And if someone did it, would it be accepted into the upstream tree?

- One possibility – this would be a separate project. Advantage is that it would decouple testing and the release process.

Rupert – a related issue – support for PPC64? There is a Bugzilla bug (OFA Bugzilla), it wouldn’t build; the RPM wouldn’t install because there is a library that is not available on PPC.

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

**Future Agenda Topics:**

- Sean’s rsockets topic

**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=7324f1bbd21440c7ae3704049190c555)

**Next regular telecon**

Next meeting: Tuesday, 8/11/15

9am-10am Pacific daylight time