Group	Requirement	Done (0–1)	Acked	Notes / How Met
MPI				
1	Message boundaries	1		MSG and tagged interfaces
2	Minimize instructions in critical path	1		Optimized APIs with control information for operations configured during initialization.
3	Zero copy	1		Data transfer interfaces are asynchronous and can support zero-copy, subject to provider hardware capabilities.
4	One-sided transfers	1		RMA interfaces
5	One-sided atomics	1		Atomic interfaces
6	Two-sided semantics	1		MSG and tagged interfaces
7	Arbitrary buffer alignment for data transfers	1		No requirement specified
8	Asynchronous progress independent of API calls			See progress discussion to determine if MPI needs are met. https://www.openfabrics.org/downloads/OFIWG/2014-05-06-ofiwg-progress.pptx https://www.openfabrics.org/downloads/OFIWG/2014-05-13-ofiwg-progress2.pptx
9	Scale to millions of communication peers	1		Introduces reliable-unconnected (reliable-datagram) model. Adds address vector class to improve address resolution performance and minimize local address data memory requirements. Allows AVs to be shared among multiple processes on a single node, and will tie into the scalable SA framework. Interface concepts are there, but underlying implementation is developing.
10	Reliable and unreliable communication	1		Multiple endpoint types are defined
11	Connectionless communication	1		Multiple endpoint types are defined
12	Specify remote RMA address	1		RMA interfaces
13	RMA write with immediate	0.5		RMA writemsg interface and fi_eq_data_entry for completions. Support is available, but not an optimized call.
14	Larger RMA write immediate data	0.5		RMA writemsg and fi_eq_data_entry support 8-byte immediate data. Iovec could be used to transfer more than 8-bytes of

				immediate data. Completion support would require compatible changes to fi_eq_data_entry.
15	Reuse short buffers immediately	1		FI_INJECT flag and 'inject' data transfers allow for buffer reuse. Provider indicates both the maximum size of single transfer and the maximum total amount of buffer space available. No restriction is placed on the provider implementation, but inline is supported.
16	Native OS polling and blocking support	1		Wait objects are selectable by application and may be retrieved for use in native calls (e.g. select/poll/pthread). For performance reasons, fabric interfaces are also defined for polling / waiting on objects.
17	Discover device, ports and their capabilities, but not tied to specific hardware models.			Proposal abstracts device and ports from application. Provider and 'domain' concepts expose application capabilities and usage requirements (for maximum performance). Discovery is built around fi_getinfo call, but operates at a higher level of abstraction than a device level.
18	SGL support	1		All operations support SGL
19	Atomic support	1		Atomic interfaces provide full set of operations and data transfer sizes.
20	Multiple consumers in a single process. Independent handles.	1		
21	Avoid collective initialization across multiple processes	1		
22	Independent process images between peers	1		
23	Separate completion order from delivery order			No intent to make this association, but see ordering discussion (TBD) to determine if MPI needs are met.
24	Support any process address region – stack, heap	1		Memory registration constraints and zero copy support limit use of stack space for data transfers – see FI_INJECT.
25	Do not require a specific wire protocol	1		Support for multiple wire protocols will be supported, including support for provider specific protocols (e.g. Intel PSM) and external protocols layered over lower-level protocols (e.g. rsockets over IB/iWarp RC QPs). Underlying protocol exposed

				through fi_info structure. Applications must adhere to any low-level protocol requirements, such as 40-byte GRH UD header, but such requirements are enforced only when that protocol is used.
26	Ability to establish connections	1		CM interfaces
27	Must grant permission for peer access to memory	1		Registration is required for remote access to local memory.
28	Clean up resources on process termination	1		Kernel requirement to reclaim any allocated resources.
29	Expose MTU for unconnected data transfers	1		Endpoints have a FI_OPT_MAX_MESSAGE_SIZE (size_t) property.
30	Control over CM timeouts	1		Use administrative file interfaces to specify CM timeout / retry values (~ /proc/sys/net/ipv4). Provide endpoint control options for an application to override defaults.
31	Support non-blocking address handle creation	1		Address vector interfaces are asynchronous.
32	Support non-blocking CM calls	1		CM interfaces are asynchronous.
33	Support non-blocking memory registration	1		Memory registration interfaces are asynchronous.
34	Specify buffer / length as function parameters – use fewer structures to minimize memory accesses	1		Optimized data transfer APIs take buffer and length as parameters.
35	Query number of send credits available	0.5		Data transfer APIs return EAGAIN if queues are full.
36	Eliminate 'queue pair' concept, and replace with send and receive channels	0.5		Queue pair is replaced with more generic 'endpoint' class. Endpoints may be send-only, receive-only, or both. An endpoint may support multiple data transfer flows. To support connection-oriented endpoints, send and receive channels may need to be tightly coupled.
37	Completion at target for RMA write	0.3		RMA and event queue interfaces support this notion. Need mechanism for provider to indicate if this is supported and to document the expected behavior. Are events at the target side associated with an endpoint or a memory region bound to an event queue?

38	Ability to determine if loopback communication is supported			Assumption is that loopback communication must be supported by providers.
39	Document what functionality must be provided, versus which is optional	0.9		Mechanism is available, but specific functionality needs to be determined. Intent is to allow providers to optionally support specific functionality. Some support may require provider specific protocols.
40	Improve ability to determine cause of errors	1		Provider specific error codes and strerror functionality are exposed.
41	Standardized high-level tag matching interface	1		Tagged interfaces
42	Standardized high-level non-blocking collective operations	0.3		Triggered operation support defined as a collective building block.
43	Standardized atomic operations	1		Atomic interfaces
44	Providers must support full set of interfaces, even if emulated	0.5		Providers are free to support all interfaces. Proprietary protocols are supported. The framework can provide emulated interfaces over device specific interfaces (e.g. libibverbs) that providers can re-use. No plans to require providers to support any specific interfaces, or to what extent they must be supported.
45	Run-time query to determine which interfaces are supported	1		The fi_info protocol_cap field indicates which interfaces are supported by a provider. Additional query functionality is provided for atomic support.
46	Direct access to vendor-specific features	1		Applications can open provider specific interfaces by name. All framework classes support provider specific interface extensions.
47	Run-time version query support	1		Version data available through query interfaces.
48	Compile-time convention for safe, non-portable code	1		FI_DIRECT allows building against a specific provider, with documented compile-time flags that a provider must set to allow highly-optimized application builds. Providers may override static inline wrapper calls and select enum values to support function inlining.
49	Direct access to vendor	1		Framework only intercepts a small number of calls. All critical calls go directly to the provider.
50	Run-time query to determine if	1		FI_LOCAL_MR domain capability flag. Long term goal to move

	memory registration is necessary			registration caches into framework.
51	Notification of forced memory deregistration (e.g. munmap)			
52	Fork support – parent process may continue to use all opened handles and fabric resources			Any effect on API?
53	For support – opened fabric resources are not shared with child processes. Child must re-initialize and open any desired resources			Any effect on API?
54	Do not require use of GRH (network specific header) with data transfers.	1		MSG interfaces allow posting of GRH headers for applications that need them, but posting is not required, and the GRH format is not specified as part of the API. The exposed low-level endpoint protocol indicates if a GRH is required or not.
55	Request ordered versus unordered delivery, by traffic type (send/receive versus RMA)	0.5		See ordering discussion (TBD) to see if MPI needs are met.
56	Allow listeners to request a specific network address	1		Endpoint creation and CM interfaces.
57	Allow receivers to consume buffers directly related to size of incoming message (e.g. slab buffering)	0.5		FI_MULTI_RECV flag adds support for slab receive buffering. Need mechanism to indicate support.
58	Aggregate completions	1		Event counters interfaces.
59	Out-of-band messaging			Need clarification. Endpoints have the concept of multiple flows, which might be useful here.
60	Non-contiguous data transfer support	0.5		Struct iovec is supported. Other formats would require extensions to the API or special interpretation of iovec data.
61	No page size restriction	1		
62	Access to underlying performance counters	0.5		Event counter interfaces. Need to verify if APIs are usable for generic purposes, such as reading performance counters, and document their usage in such cases.
63	Get/set network QoS levels	1		Endpoints getopt/setopt interfaces.
64	Atomic support for all C types	1		Atomic interfaces – checked against MPI defined types. Provider

				support is optional, but queryable.
65	Full set of atomic operation support	1		Atomic interfaces – checked against MPI operations. Provider support is optional, but queryable.
66	Query to determine if atomic operations are coherent with host	1		FI_WRITE_COHERENT flag.
67	Offset based communication – RMA target address as offset			
68	Allow application to discover if VA or offset based RMA performs better			
69	Aggregate completions per endpoint and per memory region	0.5		Event counters interfaces. Need to define/document use case for per memory region, versus per endpoint.
70	Specify remote access keys (rkeys) when registering	1		MR interfaces, FI_USER_MR_KEY capability flag.
71	Specify arbitrary sized atomic ops	0		Atomic interfaces limited to full set of C types. Need clarification.
72	Specify/query ordering of atomics			See ordering discussion (TBD) to see if it meets MPI needs.
73	Provide network topology data			Fabric class defined where topology data would go. Topology interfaces and data structures are not defined.
74	Without tag matching, need to send/receive two buffers			Tagged interfaces are defined.
75	Optional support for thread safety	1		Compile and run-time threading options, similar to MPI. fi_threading enum.
76	Support for checkpoint/restart. Allow closing stale handles that may not have a matching kernel resource.			Any effect on API?
77	No assumption of maximum transfer size			Maximum message sizes supported by provider exposed through attributes.
78	No assumption that memory translation is in network hardware			Any effect on API?
79	No assumption communication buffers are in RAM			Any effect on API? Do we need a flag to indicate that an address range is I/O mapped?
80	Support both onload and offload hardware models	1		See discussion on progress (links above) to see if MPI needs are met.

81	No assumption that API handles refer to unique hardware resources	1		Handles are abstractions, with no requirement to map to specific hardware resources.
82	Have well-defined failure semantics communicating with peers			Need to define error reporting for unexpected disconnect and unreachable unconnected peers.
Rsockets ES-API				
83	Single wait object and event queue for CM and CQ events			
84	In-band disconnect notification			
85	Associate transport resource with an fd for fstat, dup2, etc. support			
86	Fork support, even if resources must migrate from user space to kernel			
87	Chroot support			No file paths exposed in API. Administrative configuration makes use of file system.
88	Eliminate RMA address exchange – offset based transfers	-		See 67.
89	Eliminate RMA rkey exchange – user selectable key	-		See 70.
90	Target RMA write event – buffer and length	-		See 37.
91	Eliminate posting receives when only using immediate data			Need to document. May need to define local/remote EQ overflow notification.
92	Target side support for slab based receive buffer(s).	-		See 57.
93	Indicate completed send operations using count. Support different sends updating different count values.			Related to 58. Need to define usage of multiple counters.
94	Target side support to separate received data into multiple buffers (i.e. header and data), both using slab based buffering.			Related to 57. No mechanism defined for splitting received data between buffers.
95	Completion notification of partially			EQ interfaces can support this. No mechanism defined for how a

	received large data transfers.			user would configure the notification threshold.
96	Signal fd when transport is able to accept new data.			
97	Keep-alive support - optimized 0-byte transfers that are acked at target, but do not consume target resources	0.5		0-byte RMA write support possible, but may introduce extra RMA protocol header.
98	Scalable transport address resolution and storage – user-selectable unconnected transport addresses			
99	Multicast support	1		CM multicast interfaces
100	Increase immediate data size – provide mechanism to determine supported size.			Related to 14.
101	Timeout values for all CM ops	-		See 30.
102	Timeout value for reading events			
103	Ability to cancel outstanding operations	1		Endpoint cancel interface
104	Document what error codes all calls may return	0.2		
105	Use a single error return convention	1		Calls return –(fabric errno) on error.
106	Consistent error values in events	1		Error events return fabric errno, along with provider specific error code.
107	Easy mechanism to display error text	1		Related to 40. Strerrro functions defined for converting error values into text, including provider specific error codes.
108	Query status of local queues			Related to 35.
109	Support memory registration at the system level	0		Memory registration interfaces defined at a domain level. Interfaces would be usable at a wider level, but with constraints, such as requiring user selectable protection key.
110	Detect any memory alignment restrictions, if any.	-		See 7.
111	Discovery of inline data sizes	-		See 15.
112	Define required minimum SGL size			
113	Define required minimum inline size			
114	Define required minimum immediate			

	data size			
115	Define required minimum private data size			
116	Support multiple providers	1		
117	Provide full test suite – simple examples, performance tests, compliance tests (errors and min/max values)	0.1		Fabtests to evolve into full test suite.
SHMEM PGAS				
118	Scalable endpoint memory usage	1		Reliable unconnected endpoint and AV interfaces
119	Low-overhead mechanism to enumerate endpoints (i.e. ranks)	1		AV table interfaces
120	Connectionless RMA and message interfaces	1		Data transfer interfaces include connectionless operations
121	Support dynamic connection establishment	1		CM interfaces
122	Support all to all connection efficiently	0.5		CM and AV interfaces
123	Support for 'thread hot' thread safety model	1		fi_threading models
124	Scalable memory registration – user selectable rkey	-		See 70.
125	Scalable memory registration – offset based addressing	-		See 67.
126	Separate registration from page pinning, to support sparsely populated memory regions			
127	Allow memory regions to grow up/down			
128	Signaled RMA put that writes flag value to specified memory location after all RMA put data is available at target	0.5		RMA write with immediate as possible option
129	RMA operation increments counter at	1		Remote updates to registered memory regions may increment a

	receiver on completion, for scalable global communication patterns			counter
130	RMA put completion when local buffer is re-usable	1		EQ interfaces
131	RMA put completion when remote buffer has been updated	1		FI_REMOTE_COMPLETE flag
132	Blocking RMA put until local buffer is safe to reuse	1		FI_BUFFERED_SEND flag and inject calls
133	RMA put data ordering requirements: WAW, WAR, RAW	0.5		See ordering discussion (TBD) to see if PGAS needs are met.
134	Rich set of atomic operations	1		See 65.
135	Multi-element atomic operations	1		Atomic interfaces
136	Support 16, 32, 64, and 128-bit atomics	0.75		Related to 64. 128-bit not defined
137	Support high-performance 'estimated' atomic operations			
138	Lightweight aggregate completion mechanism for RMA get/put	1		See 58.
139	Notification of get completion at target that indicates read buffer may be reused			
140	Fencing operation between RMA transactions			Fi_sync call
141	Per transfer networking ordering options	0.5		See ordering discussion (TBD) to see if PGAS needs are met.
142	Larger RMA write immediate data			See 14.
143	Low-level common collective interfaces – barrier, reductions, alltoall, allgather			
144	Active message support			
OFIWG				
145	Support sharing receive buffers across multiple connections / endpoints	0.5		Receive endpoint class
146	Multicast loopback suppression; preferred per endpoint option. (See			

	IP_MULTICAST_LOOP option.)			
147	Send-only multicast support			
148	Control multicast routing and backpressure options. Detect multicast congestion.			
149	Support receiving all multicast traffic. See IP_MULTICAST_ALL option.			
150	Support promiscuous endpoints.			
151	Support flow steering capabilities			
152	Allow the completion of one request to indicate that some set of previous requests have completed.			
Oracle				
153	Signaled and silent completions			
154	Message based communication			
155	Simplex communication channels			
156	Ordered, reliable, non-duplicated messages.			
157	Timestamp part of endpoint address			
158	Optimized address change notification. Not per connection.			
159	Scaling to very large process counts (10,000s) per node.			
160	Support difference QoS levels for different data transfer flows.			
161	Provide virtual isolation between databases.			
162	Optimize resource sharing – registration, etc. – across processes using symmetric memory.			
163	Want single use RMA buffers or mechanism to cancel a pending receive buffer.			

164	Describe system NUMA architecture and mapping to fabric resources: devices, interrupts.			
165	OS supported wait mechanisms. Cannot poll to drive state.			
166	Support for triggered operations – across processes and/or endpoints.			
167	RMA write completion notification at target.			
168	Completion of RMA writes to persistent memory target.			