If a process or thread attached to a node fails it is generally up to the application to recover from this failure. MCAPI provides timeouts, as endpoint attributes as well as for the `mcapi_wait()` and `mcapi_wait_any()` functions and provides the `mcapi_cancel()` function to clear outstanding non-blocking requests (at the non-failing side of the communication). It is also possible to reinitialize a failed node, by first calling `mcapi_finalize()`.

3.11 MCAPI Buffer Management Philosophy

MCAPI provides two APIs for transferring buffers of data between endpoints: MCAPI messages and MCAPI packet channels. Both APIs allow the programmer to send a buffer of data on one node and receive it on another node. The APIs differ in that messaging is connectionless, allowing any node to communicate with any other node at any time, with pr message, priority. Packet channels, on the other hand, perform repeated communication via a connection between two endpoints.

The APIs for sending a buffer via messaging or packet channels are very similar. Both APIs have a send method that allows the programmer to specify a buffer of data with two parameters: (void*, size_t). The programmer may send any data buffer they choose. There is no requirement that the buffer be allocated by MCAPI or returned to MCAPI after the send call completes.

The messaging and packet channels APIs differ in their handling of received buffers. The messaging API provides a "user-specified buffer" communications interface – the programmer specifies an empty buffer to be filled with incoming data on the receive side. The programmer can specify any buffer they choose and there is no requirement to allocate or release the buffer via an MCAPI call.

On the other hand, packet channels provide a "system-specified buffer" interface – the receive method returns a buffer of data at an address chosen by the runtime system. Since the receive buffer for a packet channel is allocated by the system, the programmer must return the buffer to the system by calling `mcapi_pktchan_release()`.

MCAPI data buffers may have arbitrary alignment. However, MCAPI does define two macros to help provide buffer alignment in performance-critical cases. Use of these macros is optional and can be defined with no value ("no-op"); implementers are required to handle send and receive buffers of arbitrary alignment, but can optimize for aligned buffers.

MCAPI implementations define the following two macros:

1. `MCAPI_DECL_ALIGNED` : a macro placed on static declarations in order to force the compiler to make the declared object aligned. For example:

```
mcapi_int_t MCAPI_DECL_ALIGNED my_int_to_send; /* See mcapi.h */
```

2. `MCAPI_BUF_ALIGN` : a macro that evaluates to the number of bytes to which dynamically allocated buffers should be aligned. For example:

```
memalign(MCAPI_BUF_ALIGN, sizeof(my_message));
```

MCAPI does not provide for any maximum buffer size. Applications may use any message size they choose, but implementations may specify a maximum buffer size and may fail with `MCAPI_ERR_MEM_LIMIT` if the system runs out of allocatable memory. Thus, the maximum size of a message in any particular implementation is limited by an implementation specified size or the amount of system memory available to that implementation.