The ExaNIC Sockets acceleration library allows applications to benefit from the low latency of direct access to the ExaNIC without requiring modifications to the application. This is achieved by intercepting calls to the Linux socket APIs.

While ExaNIC Sockets should be compatible with most applications using Linux socket APIs, there are some cases where programs may not work as expected. Feedback and bug reports would be greatly appreciated (contact us at support@exablaze.com).

1. Known issues and limitations

- Each thread that calls a blocking I/O call - e.g. select(), poll(), epoll_wait(), recv(), read() or accept() - will spin waiting on data. This normally provides optimal latency but can induce performance problems if there are more threads than available CPUs. Other blocking modes will be provided in the future.
- If a socket is bound to a wildcard address (`INADDR_ANY`) or to a multicast address, it will only receive packets that arrive on ExaNIC interfaces when run with the acceleration library.
- Connecting to an accelerated socket from the same host is not supported (for example, if a socket is bound to 192.168.1.1:80, then it is not possible to connect to 192.168.1.1:80 from the local host).
- Transmitted multicast datagrams are not looped back to local sockets.
- The MSG_WAITALL flag to recv() is not currently supported (to be resolved).
- No support for recursive addition of epoll file descriptors to epoll sets.
- No support for IP fragmentation.
- Sockets may not be correctly maintained across `fork()` or `execve()`.
- Sockets cannot be transferred to other processes with `sendmsg()`.

2. Software installation

Build the ExaNIC driver and libraries as per the ExaNIC Installation and Configuration Guide. ExaNIC Sockets is built and installed as a standard component, and the exasock kernel module is loaded automatically when an ExaNIC interface is brought up.

3. Usage

First ensure that the application works without ExaNIC Sockets. All IP addresses should be configured as if you were running the application through the normal Linux network interface corresponding to the ExaNIC.

Then, to accelerate the application, simply prefix it with the `exasock` command. For example, to run the UNIX netcat (`nc`) utility to listen for UDP datagrams on port 1234:

```
$ exasock nc -u -l 1234
```

Another simple example application that receives and sends UDP multicast datagrams is located in `examples/exasock/multicast-echo.c`. Note that this is a normal Linux sockets application that can be run either with or without the ExaNIC Sockets acceleration library.

Sometimes it can be difficult to determine if the kernel bypass is functioning correctly. Setting the `EXASOCK_DEBUG` environment variable prints extra debugging information that can help. For example:

```
$ EXASOCK_DEBUG=1 exasock nc -u -l 1234
exasock: enabled bypass on fd 4
```

In this case, the message '`exasock: enabled bypass on fd 4`' indicates that kernel bypass has been enabled for the socket associated with file descriptor 4.

4. Tips for best performance

- Wherever possible, do not mix accelerated sockets with non-accelerated sockets and other file descriptors in `select()` and `poll()` calls.
- For the best possible performance, pin threads to CPU cores in the CPU socket directly connected to the ExaNIC.