

**NAME**

libcxgb4\_udp – a User Mode UDP (UM-UDP) library for the cxgb4 device.

**SYNOPSIS**

```
#include <chelsio/cxgb4_udp.h>
```

**DESCRIPTION**

This API provides an asynchronous RDMA Verbs-like interface allowing zero-copy low latency UDP communications. It is intended to be used by middleware to implement a full user mode UDP sockets interface, as well as by custom applications needing low latency / high packet rates. The API provides a fastpath QP interface for sending and receiving non-fragmented unicast and multicast UDP datagrams, and a slowpath QP for receiving multicast IP fragments needing reassembly. It is assumed the middleware using this interface will also use host UDP sockets to handle sending of UDP datagrams needing fragmentation, as well as reception of unicast fragmented UDP datagrams. The combination of the cxgb4\_udp queue pairs and one or more host UDP sockets can be used to implement a full user mode udp socket.

The cxgb4\_udp API is a thin layer sitting on top of Linux RDMA Verbs, and leverages the existing Linux RDMA API to support:

- Registration, pinning, and mapping of user memory to enable direct IO from adapter to user memory.

- Neighbor address resolution (ARP).

- The IBV\_QPT\_RAW\_ETY QP type to send/recv raw ethernet frames.

- Completion queues and channels allowing asynchronous polling and blocking for IO completions.

T4 hardware filters are used to route only the correct raw ethernet frames to a given UDP queue pair. A udp\_qp can send/receive only non fragmented unicast and multicast packets for the bound UDP port. The hardware also provides Ethernet CRC, IP cksum, and UDP cksum verification and generation.

**THEORY OF OPERATION**

To setup a UM-UDP endpoint, an application would first allocate a udp device using **udp\_alloc\_dev(3)** to find the RDMA device associated with the desired IP address, and obtain a context to that device. If no T4 devices are found at that IP address, then **udp\_alloc\_dev(3)** returns ENODEV. Once the device is allocated, the application should allocate a cq completion channel via **ibv\_create\_comp\_channel(3)** and then create a CQ using **ibv\_create\_cq(3)**. This cq will be used for processing slow-path ingress multicast fragments. Memory for zero-copy low latency IO operations (aka fastpath IO) can also be allocated and registered with the device at this time by calling **ibv\_reg\_mr(3)**.

Next the application calls **udp\_start\_dev(3)** to start receiving slow-path multicast fragments. Fragments are obtained by calling **udp\_poll\_frag(3)** to get a copy of each ingress fragment. Since the application has full control over the cq, it can spin poll via **udp\_poll\_frag(3)** or block using **ibv\_req\_notify\_cq(3)**, **ibv\_get\_cq\_event(3)** and **ibv\_get\_ack\_cq\_event(3)**. To setup the fast path udp queue pair, the application next calls **udp\_create\_qp(3)** passing in the desired recv and send cqs, the udp port number, and the qp send and recv depths. Once the udp qp is created, the application can send non-fragmented unicast or multicast datagrams using **udp\_post\_send(3)**. The application must also keep receive buffers available for receiving non fragmented unicast and multicast datagrams using **udp\_post\_recv(3)**. Completion processing for the fastpath udp qp also uses standard RDMA Verbs cqs and completion channels, but a new poll method, **udp\_poll\_cq(3)**, is needed to poll for both send and recv completions. These send and recv functions handle adding/removing the UDP/IP/Ethernet headers. So the application need only post ULP payload.

The application is expected to also allocate a host udp socket and bind it to the same udp port number as the udp qp. This is needed to handle the remaining slowpath operations: sending/receiving datagrams needing fragmentation, as well as off-link sends.

The application can join IP multicast groups by calling **udp\_attach\_mcast(3)** and leave groups by calling **udp\_detach\_mcast(3)**. These multicast functions simply enable reception of that particular IP multicast address on the udp qp. It is up to the application to support IGMP for off-link multicast.

**SEE ALSO**

**udp\_alloc\_dev(3), udp\_dealloc\_dev(3), udp\_start\_dev(3), udp\_stop\_dev(3), udp\_poll\_frag(3),  
udp\_create\_qp(3), udp\_destroy\_qp(3), udp\_post\_send(3), udp\_post\_recv(3), udp\_poll\_cq(3)  
udp\_attach\_mcast(3), udp\_detach\_mcast(3)**