XGE 10Gb Ethernet

UDP Direct Transfer Protocol

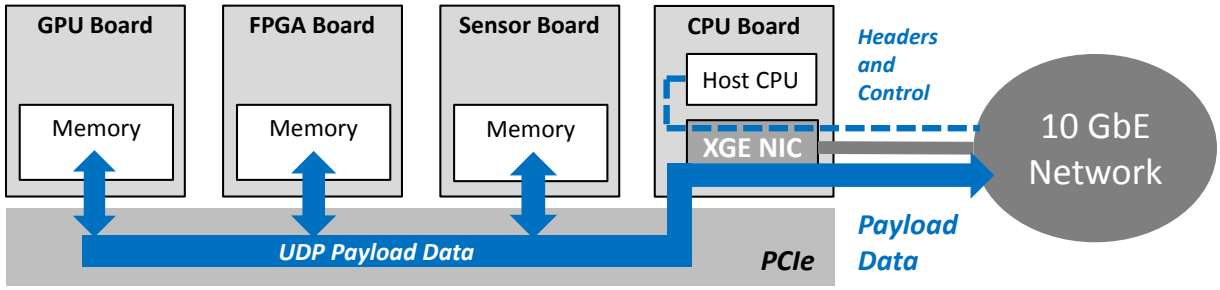
Abstract

Critical I/O's UDP Direct stream transfer protocol is a highly efficient method of moving block UDP data over standard 10 GbE networks, using a completely standard UDP protocol. UDP Direct allows very large blocks of UDP payload data to be sent and/or received directly to/from a user's applications level buffers that can be located anywhere in PCIe address space, including buffers that may be located on separate cards such as FPGA cards, DSP cards, GPU cards, and others, with no host CPU data copies. Full 10 GbE line rate data transfer is easily achieved, with very low host CPU loading

Revised 7/18/2014

Critical I/O XGE Ethernet UDP Direct Transfer Protocol

Critical I/O's UDP Direct transfer protocol and XGE NIC hardware provide a highly efficient method of moving block UDP data over standard 10GbE networks, using completely standard UDP as the on-the-wire protocol. With a typical i7 CPU hosting the CIO UDP Direct driver, full 10GbE line rate sends and receives can be achieved using less a 5% loading of one CPU core.



The UDP Direct stream mode of operation can be used concurrently with general purpose 10GbE netwok traffic using the normal network stack. The XGE NIC hardware, firmware, and driver software support simultaneous usage for UDP direct stream transfers and standard networking.

As implied by the name, at UDP Direct mode applies to UDP traffic streams only. A "UDP stream" is defined as a flow of data (a series of UDP datagrams) transferred between two UDP "endpoints", where each endpoint is defined by IP address and UDP port. For example, a connection between [192.168.5.1: 1005] and [192.168.5:1025] would be a stream. (IP addresses and UDP ports can also be wildcards).

The XGE NIC chip hardware and firmware inside, along with the Critical I/O UDP Direct driver, have the ability to "steer" incoming UDP traffic to a specific set of receive buffers associated only with that specific stream, and also have the ability to strip off the various Ethernet/IP/UDP packet headers prior to writing the payload data into those buffers. The set of receive buffers can be located anywhere in PCIe address space, such as memory within an external GPU or FPGA card. This is very significant because data will be deposited by the XGE NIC hardware right where it is needed, with no host CPU data copies and no network stack overhead.

As an example, a user could define a 1 MB buffer located within an external GPU memory. A single call to the CIO driver is made to associate this "big" buffer with a specific stream. The driver and NIC firmware will divide the 1MB buffer into individual datagram buffers that are right size to receive the payload portion of the incoming datagram stream. Only when the 1MB buffer is completely filled with incoming datagrams (or it times out) will the XGE NIC generate an interrupt which will then result in the driver providing a user RX completion notification. Only a single driver call is needed to set up to receive the 1MB of data directly into GPU memory, and then provide the completion to the user application when all of the data has been received. Furthermore, the packet headers have been stripped off which results in the payload data being packed contiguously into memory.

The stream receive capability is quite flexible with respect to the number and size of buffers that can be queued, and the receive driver API calls can be blocking or asynchronous.

UDP Direct provides an equivalent "stream send" capability that functions nearly identically to the receive capability. For stream sends, the user defines a buffer anywhere in PCIe address space of arbitrary size. A single call to the CIO driver will result in the driver and NIC firmware initiating a send of the full buffer, with the NIC automatically breaking the buffer up into as many identically sized UDP datagrams as are needed to send the full user buffer. A single completion notification is generated when the full buffer has been sent.

Note that while the other side of the interface can also be using the stream mode, it does not have to. It can also just send (or receive) data via standard socket calls, provided the datagrams are the correct size.

There are several restrictions that must be observed. For receive data to be packed contiguously in the user's "big" receive buffer, the sender must send datagrams of consistent size, and the datagram size must match the size that is defined on the receive side. Fragmented datagrams are not supported for either sends or receives. In the typical case where the user defines the operation for both the send side and receive sides of the interface these restrictions do not present a limitation.

UDP Direct API

The UDP Direct API provides the user application interface to send and receive streams of UDP datagrams. The functions available within this API are:

xel_init	- Initialize the user level library
xel_end	- Cleanup the user level library
xel_udp_smsend_setup	- Set up a UDP stream for sends
xel_udp_smsend_multi	- Perform a UDP stream send
xel_udp_smsend_close	- Close a UDP send stream
xel_udp_smrecv_setup	- Set up a UDP stream for receives
xel_udp_smrecv_multi	- Perform a UDP stream receive
xel_udp_smrecv_close	- Close a UDP receive stream

Many of the API routines discussed here use the xe_buflist data structure to transfer data buffers between the application and the driver. In the following sections we discuss this data structure and each of the API routines.

The API defines a *xe_buflist* data structure as the application data interface. It is shared by both the driver and the application and is a container for transferring buffer information to and from the driver during send and receive operations. The *xe_buflist* structure is quite flexible in the variety of streaming sends and receives it supports . In simplest use, it merely points to a single large send/receive buffer, and defines the send/receive datagram size. In more complex usage, an array of *xe_buflist* entries can point to multiple large buffers, or to a sequence of individual datagram send/receive buffers.

Error Detection and Reporting

Timeouts - A timeout value can optionally be supplied when a multi-datagram receive operation is initiated. If the timeout value is reached the receive operation will be terminated and a completion generated indicating the amount of data, if any, that has been received.

Size Errors - Datagram sizes for each received datagram within a block can optionally be verified.

Missing and Out of Order Data Errors - Datagram receive order can optionally be verified in many use cases. The detection method relies on the IP Identification field to verify datagram receive order. As a result it is only applicable to cases where the UDP data sender(s) supplies datagrams with a uniformly incrementing IP Identification field. XGE UDP Direct stream sends will always have a uniformly incrementing ID field for reach stream. If sending from a system using a generic NIC and/or network stack, this generally means that the sender must not send any IP datagram traffic other than the UDP stream traffic. If sending from a hardware device such as an FPGA, the FPGA hardware and software simply need to be designed to send datagrams with the proper IP ID sequences.

Error Reporting – In the case of size errors, or missing or out-of-order data, detailed information can optionally be supplied with each receive completion that indicates the location and size of any missing or out of order data.

