CRITICAL I/O APPLICATION NOTE

Critical I/O XGE 10/40Gb Ethernet

UDP Direct for Highly Efficient GPU Data Transfer

Abstract

The use of Graphics Processing Units (GPUs) as massively parallel offload processors is growing rapidly. With this comes the challenge of efficiently moving large amounts of data to and from the GPU internal memory without unnecessarily consuming a host processor's memory or PCIe bandwidth. Critical I/O's UDP Direct stream transfer protocol can provide a highly efficient method of moving block Ethernet/UDP data over standard 10 or 40 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 GPU internal memory without passing through host memory, resulting in a dramatic improvement in data transfer efficiency.

CRITICAL I/O

© 2017 Critical I/O, LLC All Rights Reserved

Revised 4/4/2017

UDP Direct for Highly Efficient GPU Data Transfer

The processing power of GP GPUs can provides many advantages in modern embedded processing systems. A single GPU can offer a multi-TFLOP processing capability. To achieve this same level of processing throughput would require dozens of general purpose CPUs

With this immense processing capability comes the need to move large quantities of data efficiently to and from GPU memory, often coming from a sensor system. Traditionally this is done in a two-step process, where the data is first 'staged' in a general purpose host processor's memory, then the GPU is commanded to DMA the data from the processor memory. Especially in cases where this sensor data is received over Ethernet, this staging processing places a significant burden on the host processor and its memory subsystem. Fortunately, current generation GPUs support direct memory access (DMA) from external devices. This means that a device external to the GPU, and in fact potentially external to the host processor system entirely, can write (DMA) data directly into GPU memory. This is far more efficient than the typical multi-step staging processing. nVidia calls this specific GPU technology *GPU Direct RDMA*, while AMD calls a very similar feature *DirectGMA*.

As an example system, consider the simple application shown below. Sensor data must be moved from FPGA(s) on a sensor subsystem into GPU memory. The required sensor data rate for this example system is 4 GByte/s.

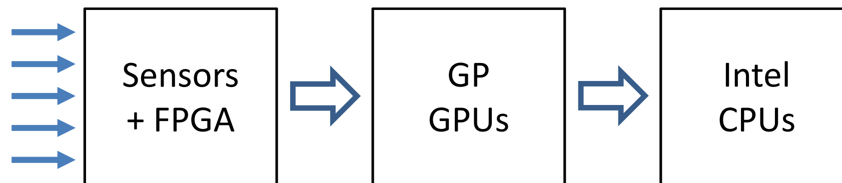

Figure 1 - A processing flow that leverages GPGPUs

Now consider the use of 10 or 40 GbE Ethernet as the transport from Sensor/FPGA to the processing system. The 4 GByte/s data rate can easily be accommodated using four 10GbE links, or a single 40 GbE link. A typical implementation using UDP as the transport would result in the data flow shown below. The UDP traffic is received and placed into buffers in processor memory (which with a standard network stack results in an extra data copy). Then data must then be DMA'd by the GPU from host memory to GPU memory which can be surprisingly CPU intensive as in many cases the host memory must be "pinned" and "unpinned" for external GPU DMA access.

Ethernet can be a good choice as a sensor interface, especially in cases there the sensor is either a smart sensor, i.e. it includes a CPU, or in cases where the sensor includes a FPGA. For the smart sensor case, the Ethernet interface implementation is obvious – it is simply a 10/40 Gb Ethernet NIC that is managed by the host CPU. The CPU can run a traditional network stack - potentially augmented with the UDP Direct API for enhanced performance and determinism as is described later in this paper.

In the FPGA case, a combination of readily available Ethernet MAC/PHY IP and relatively and a relatively simple UDP Direct send state machine logic can be used to implement a straightforward and efficient Ethernet/UDP send interface.

© 2017 Critical I/O, LLC All Rights Reserved

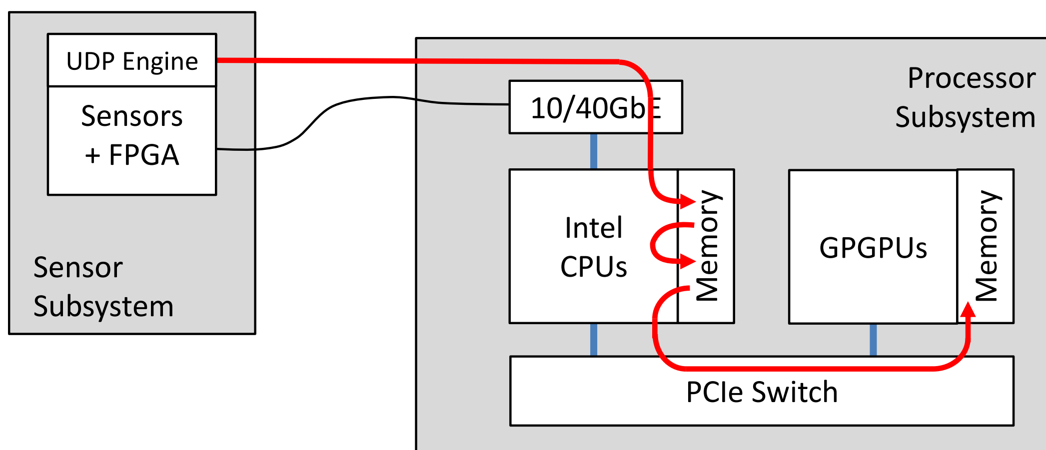

Figure 2 - Receiving sensor data using conventional UDP requires staging data in processor memory

As can be seen by looking at the CPU memory traffic pattern shown in figure 2, the 4 GByte/s data stream will actually consume four times that rate in processor memory bandwidth, for a total of 16 GByte/s of memory bandwidth consumed – clearly very significant. This is in addition to also consuming one to two processor cores just in managing the UDP receives. Combined, this represents a significant amount of valuable memory and processing resources. Obviously, it would be highly advantageous if the UDP sensor data could bypass processor memory and instead be placed directly in GPU memory.

Critical I/O has implemented an enhancement to the standard UDP protocol called UDP Direct. UDP Direct is exactly like standard UDP, except that it supports the delivery of UDP *payload* data through a 10/40 GbE NIC to any PCIe accessible memory, including GPU memory, as illustrated in figure 3. This means that UDP payload data can be delivered directly to where it is needed without needlessly consuming host processor memory, compute, and potentially PCIe bandwidth.

While the use of traditionally UDP may cause some concerns with respect to potential missing or out-of-order receipt of datagrams, UDP Direct augments traditional UDP with additional mechanisms that will detect and report these conditions. These mechanisms are described in more detail later in this paper.

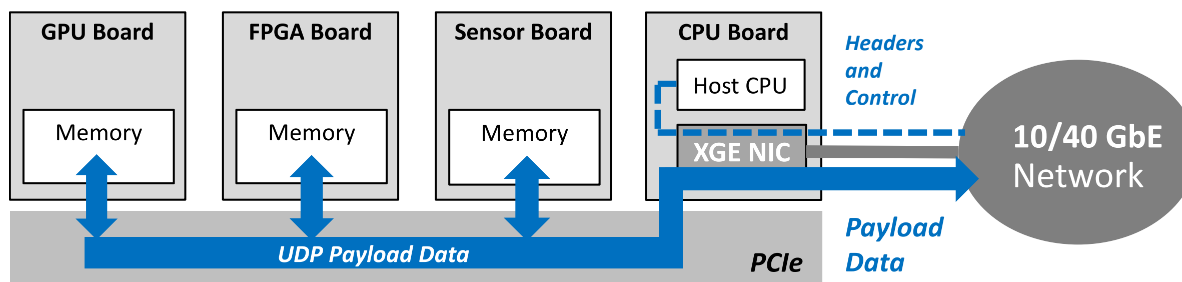

Figure 3- UDP Direct bypasses processor memory and places data directly into any PCIe accessible memory

In figure 4, we consider the same system as we looked at earlier in figure 2, but now leveraging UDP Direct. The Ethernet/UDP sensor data flow now completely bypasses host memory and data is moved directly from the 10/40 Gb Ethernet NIC directly to GPU memory. Not only does this remove the very significant loading from host memory, it also results in a substantial reduction in latency.

© 2017 Critical I/O, LLC All Rights Reserved

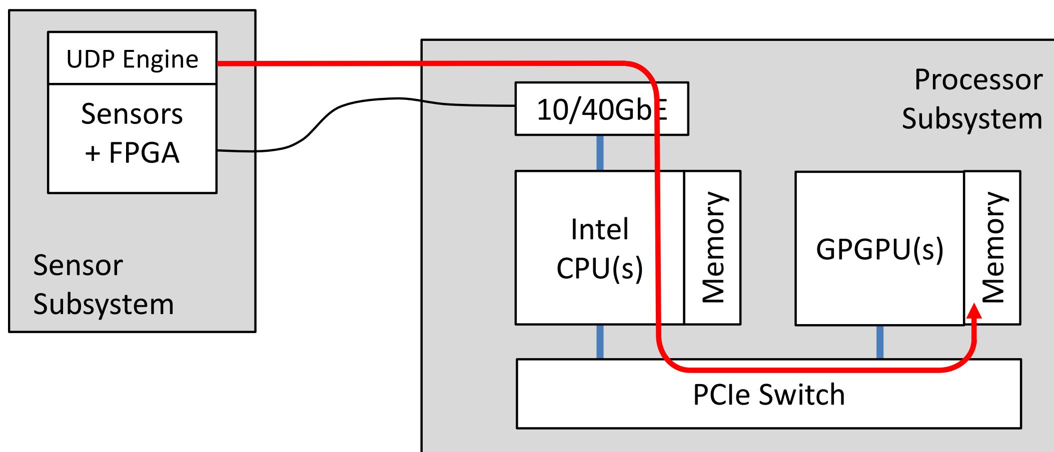

Figure 4 - Receiving sensor data using UDP Direct, completely bypassing host memory

In figure 4, the sensor data does not pass through CPU memory (and thus does not consume CPU *memory bandwidth*) but the data stream does still consume CPU *PCIe bandwidth*. However, most general purpose processor boards include on-board PCIe switches, and in such cases it may be possible for the UDP payload data to bypass the processor completely, as is shown in figure 5.

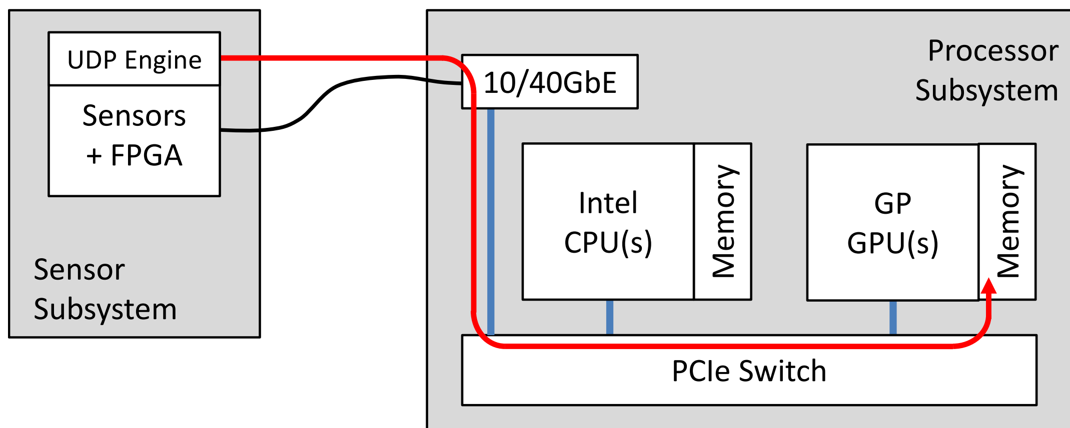

Figure 5 - PCIe switching can eliminate all data movement through CPU

A Sensor FPGA Implementation of UDP Direct

Sensors subsystems often include FPGA that perform some initial signal processing algorithms. These FPGAs can be an ideal candidate to host a UDP Direct Ethernet interface. As UDP Direct builds on a completely standard UDP implementation, and UDP itself is an extremely simple protocol, a UDP Direct FPGA implementation is also quite simple. An example implementation is shown in figure 6. Standard 10/40 GbE MAC and PHY FPGA Intellectual Property (IP) is readily available, often directly from the FPGA supplier. This "canned" IP is augmented with a relatively simple UDP Direct state machine along with a DMA engine.

While this particular implementation assumes that that the data that is to be sent is staged in external DDR memory, the data could just as easily be hosted in a FIFO or other on-chip memory. The state machine coordinates reading the data from memory, dividing the stream of data into fixed sized blocks each

© 2017 Critical I/O, LLC All Rights Reserved

representing a UDP datagram payload. The state machine then prepends Ethernet, IP, and UDP headers to each data block, forming a sequence of complete Ethernet/UDP packets that are forwarded to the 10/40 GbE IP for transport.

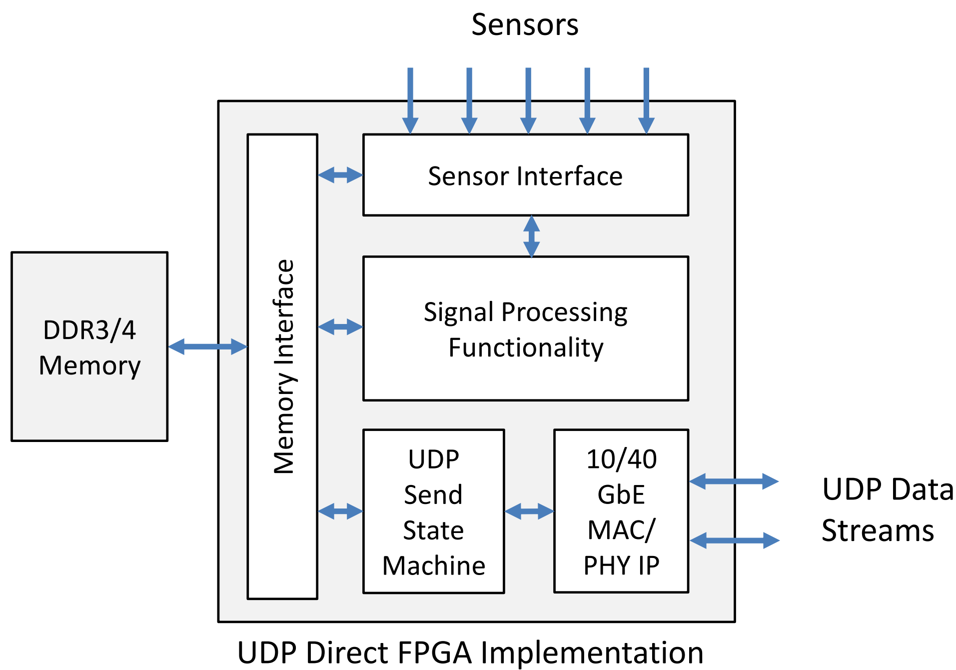

Figure 6 – An example UDP Direct Sensor FPGA Implementation

UDP Direct GPU Measured Results

The table below shows some measured data for several test cases involving moving Ethernet/UDP data to and from a GPGPU. The host processor used for this testing as an Intel Xeon-D, the Ethernet NIC was a prototype Critical I/O 40 GbE unit, and the GPU was an nVidia Pascal architecture device. All PCIe connections were PCIe Gen3 x8, with each PCIe link thus able to support up to a ~6.5 GB/s data rate.

Data is included for both sending and receiving, using two different approaches. The first approach uses conventional UDP sends or receives, with data staged in host processor memory, as was shown in figure 2 earlier. The second approach leverages UDP Direct to bypass host memory, as was illustrated in figure 4 earlier. The table shows the measured data rates as well as the associated host CPU loading (expressed as the percentage of a single CPU core that is consumed). The host CPU memory loading is shown as well, along with a "Relative Efficiency" metric which is defined as [Data Rate] / [CPU Loading]. This efficiency metric can be used to compare the relative efficiency of the two data transfer approaches.

As can be seen from this data, leveraging standard UDP completely consumes one processor core as well placing a very large load on CPU memory. The UDP Direct implementation places virtually no load on the processor, and no load at all on CPU memory. The result is that UDP Direct can be as much as 20 to 30 times more efficient than standard UDP as measured by normalized processor loading and data rate.

Though data is shown for UDP only, the use of standard TCP (instead of UDP) yields virtually the same result.

© 2017 Critical I/O, LLC All Rights Reserved

Table 1: Measured results comparing standard UDP and UDP Direct for GPU data transfer

Transport Mode	Measured Rate (MB/s)	Single Core CPU Loading (percent)	Relative Efficiency (MBpS/Load%)	CPU Memory Bandwidth Consumed (MB/s)
Standard UDP Receive, then GPU DMA	1982	93	21	7928
GPU DMA, then Standard UDP Send	4430	80	55	17720
UDP Direct Receive to GPU Memory	4931	8	616	0
UDP Direct Send from GPU Memory	4187	4	1047	0

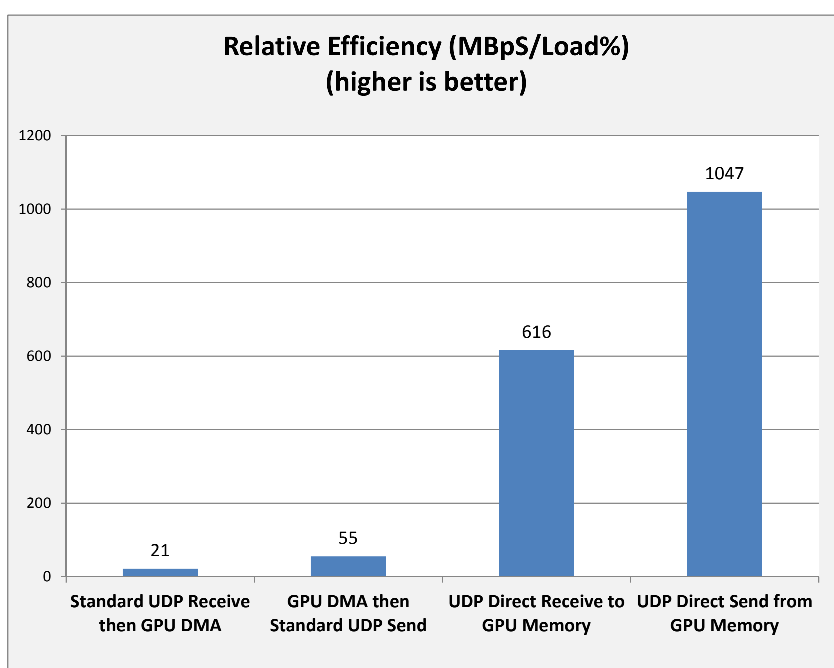

Figure 7 – Relative efficiency of standard UDP vs. UDP Direct for GPU data transfer

Additional Details of UDP Direct

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 Ethernet network 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.

Key to the implementation of UDP Direct is complete bypassing of the normal operating system network stack as is shown in figure 8. Instead, a UDP Direct API is used to initiate sends or receives of data, and the

© 2017 Critical I/O, LLC All Rights Reserved

lightweight UDP direct driver implementation communicates directly with the 10/40 GbE XGE NIC, fully leveraging the large block send/receive offload capability of the Critical I/O XGE NIC hardware and firmware.

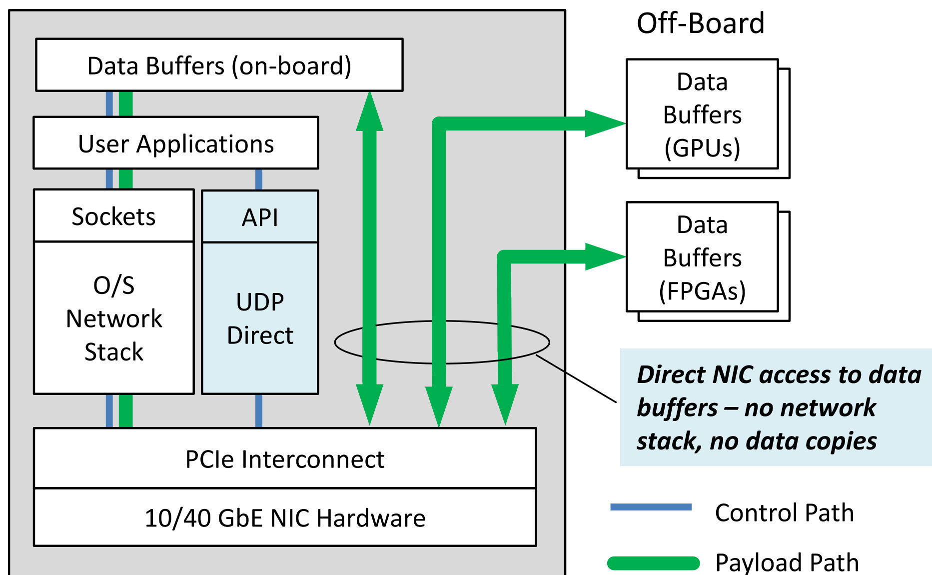

Figure 8– UDP Direct operates in parallel with normal O/S network stack

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 Critical I/O XGE NIC hardware and associated firmware, 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 API 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.

© 2017 Critical I/O, LLC All Rights Reserved

UDP Direct also provides a "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                     - Terminate 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
```

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.

© 2017 Critical I/O, LLC All Rights Reserved