XGE 10Gb Ethernet Performance

VxWorks 6.x, Linux 2.6.x

Abstract

The XGE 10G series of 10 GbE NICs provides dual port 10 Gigabit Ethernet (10GbE) connectivity for embedded systems with the high performance characteristics that are essential for data intensive real-time systems, yet maintaining 100% interoperability and compatibility with standard Ethernet infrastructures.

The drivers support three usage models: standard networking APIs, a high performance UDP Stream API, and a low latency RDMA API.

This paper provides actual measured data transfer rates and CPU loading for the XGE 10G running under VxWorks 6.x and Linux 2.6.x for a variety of modes of operation, on a variety of platforms.

Revision 4/10/2014

XGE 10Gb Ethernet Performance

VxWorks 6.x, Linux 2.6, 3.x

The XGE 10G series of 10 GbE NICs provides dual port 10 Gigabit Ethernet (10GbE) connectivity to embedded systems with the high performance characteristics that are essential for data intensive real-time systems, yet maintaining 100% interoperability and compatibility with standard Ethernet infrastructures.

The XGE 10G provides a balanced architecture which offers hardware acceleration for bulk data transfers with the flexibility of a programmable protocol processor to significantly improve network performance. It reduces the CPU cycles and burden required for 10 GbE networking, maximizing I/O bandwidth without sacrificing host CPU efficiency.

This paper summarizes the data transfer rates and CPU loading of the XGE 10G running under VxWorks 6.x and Linux 2.6.x for a variety of modes of operation, on a variety of platforms. TCP and UDP performance of more than 850 MB/s under VxWorks and 1200 MB/s under Linux have been measured.

XGE 10G Driver Models

The Critical I/O XGE 10G VxWorks and Linux Drivers allow user access to the 10GbE network interface through two different methods; using the standard VxWorks or Linux sockets API, or using a special high performance UDP Streaming API. The standards socket API accesses the NIC through the VxWorks/Linux network stack, similar to a normal NIC driver, while the UDP Streaming and RDMA models completely bypass the VxWorks/Linux sockets layer, instead using specialized APIs that directly accesses the XGE 10G hardware.

Standard Sockets API Model

The standard socket API model connects the XGE 10G driver through the VxWorks or Linux sockets interface. This allows new and existing user developed socket applications and standard network applications like FTP, Telnet, NFS etc. to make use of the XGE interface. Network performance and CPU loading is excellent, but rates are limited somewhat due to the interaction of the XGE 10G hardware with the VxWorks or Linux O/S.

For VxWorks, two modes of operation are also available using the standard sockets model. The *Moderated* mode provides very good network performance combined with lowest CPU loading through the use of interrupt moderation and coalescing techniques. The tradeoff is slightly lower peak performance, and slightly higher transfer latencies. The *Non-Moderated* mode focuses on achieving the highest possible data rates and the lowest possible transfer latencies, but at the expense of higher CPU loading.

Streaming UDP API Model

UDP Streaming provides a high performance data transfers models which leverage the offload capabilities of the XGE 10G hardware. As the standard BSD Socket datagram send/receive API is very limited, access to the UDP streaming functionality is provided via a specialized UDP streaming send/receive API. This specialized API provides very high performance UDP sends and receives with low host CPU loading.

For sends, the UDP streaming interface is used to send application supplied blocks of data as a stream of UDP datagrams, with the datagram size being a user specified value. Datagrams must be sized to fit within the current Ethernet frame size. The XGE offload hardware will break the application supplied blocks of data into a sequence of UDP datagrams, which relieves the host processor from the overhead of doing multiple individual datagram sends. Thus the application may pass very large blocks of data to the UDP streaming API to be sent, with no CPU involvement needed to perform that datagram sends, other that the initial send setup.

For receives, the application provides a large data buffer to the UDP streaming API that is to be filled with received datagrams for a defined IP/port. The offload hardware will fill the buffer with received datagrams. The application may pass very large receive buffers to the UDP streaming API to be filled, with negligible CPU involvement required to fill the buffers with data after the initial receive setup. The data stream is delivered to the application via a series of UDP datagrams that are written directly into application data buffers after stripping off the datagram header information.

RDMA API Model

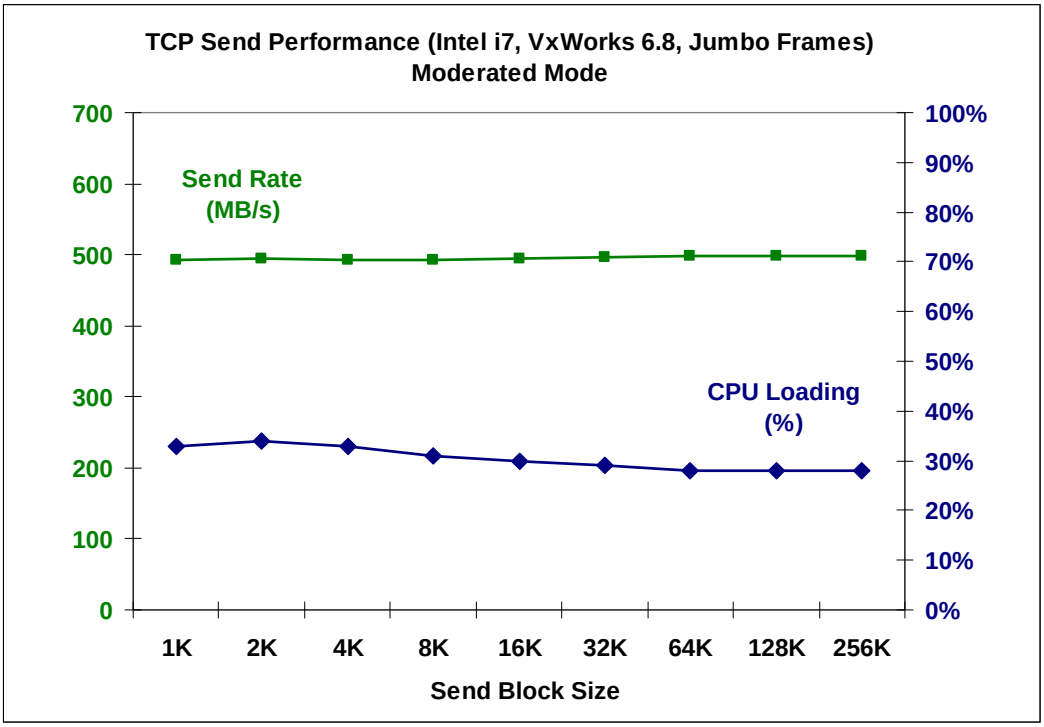
Remote Direct Memory Access (RDMA) is a capability that allows processor to processor data movement directly between application buffers with minimal CPU involvement. The RDMA API provides functionality to define local RDMA buffer regions which remote RDMA nodes can read and write without local CPU involvement, either directly or using RDMA based messaging.

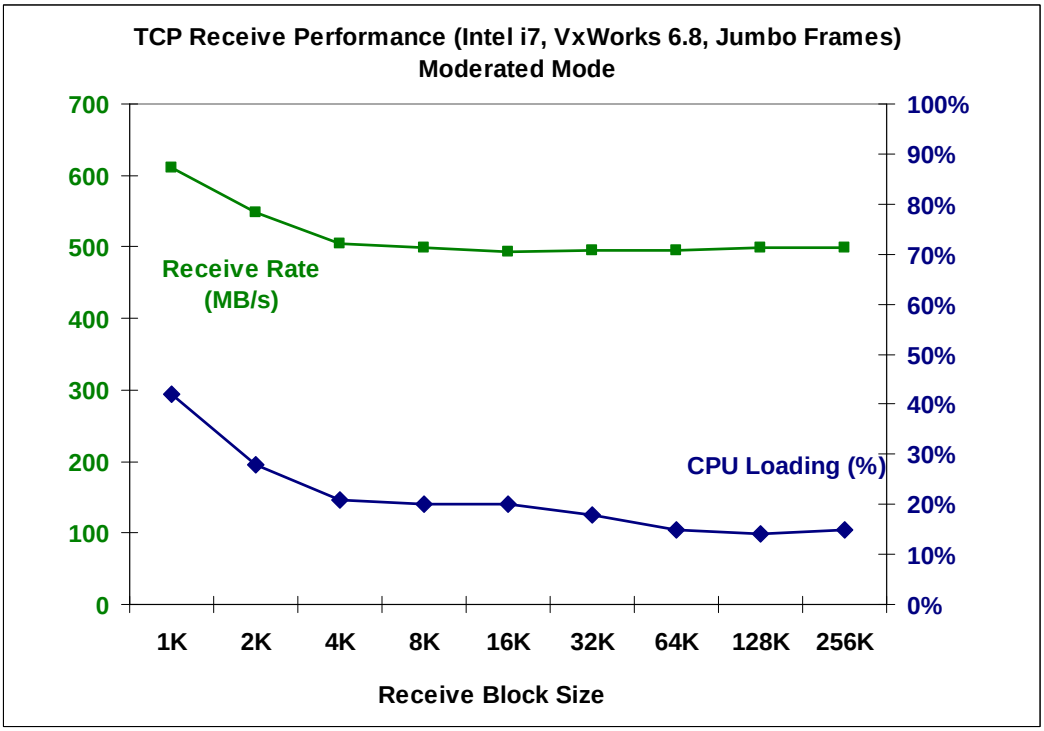
Performance Measurements

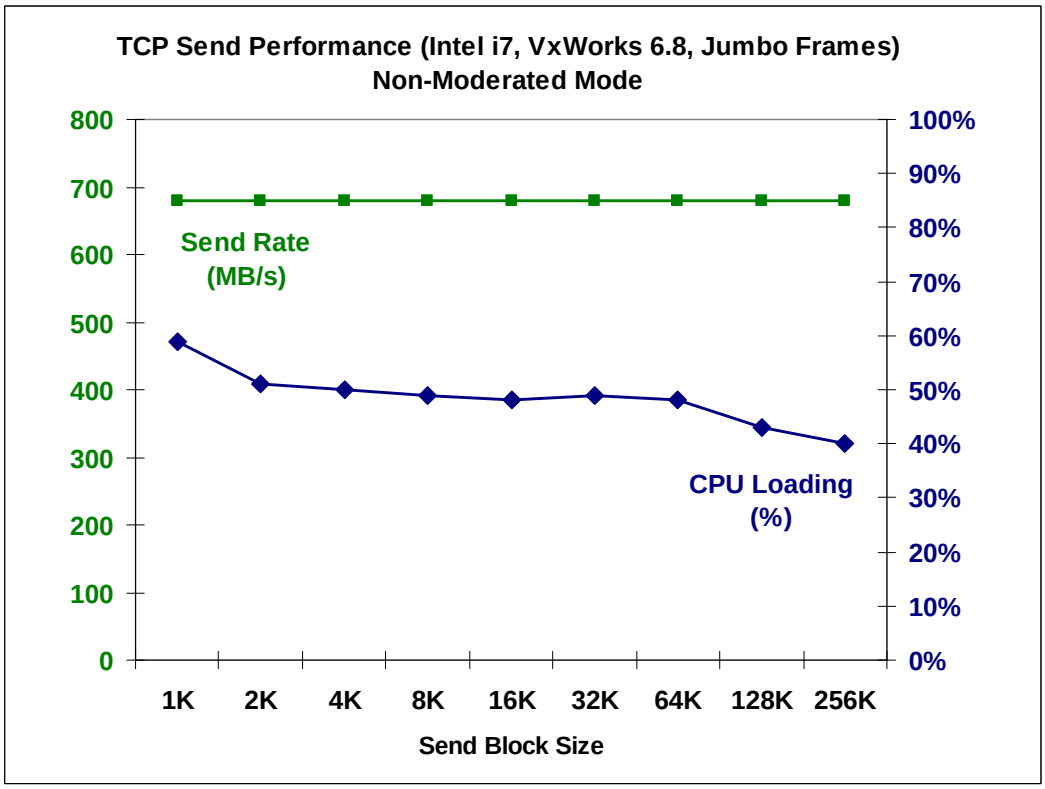
VxWorks Performance Results -- Standard Sockets API

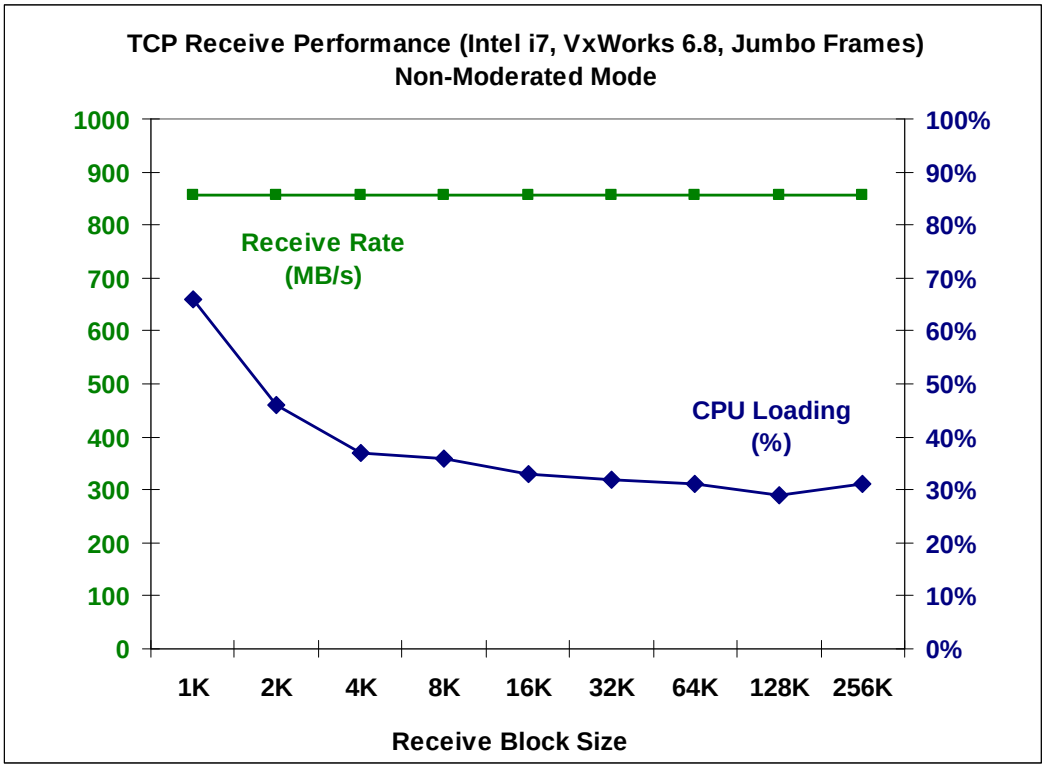
The following four charts show representative performance performing TCP sends and receives, accessing the XGE 10G through the standard VxWorks socket API. The charts show data rate in green (squares, upper lines), and CPU loading in blue (diamonds, lower lines), for a variety of send or receive block sizes. This data was taken using an embedded class processor base-board to host an XGE 10G XMC. The processor board used a dual-core Intel i7 processor, running VxWorks 6.8 SMP. Standard 9K jumbo frames were used.

The first two charts show results performing TCP sends and receives using the Moderated (interrupt coalesced) mode, while the last two charts show results TCP sends or receives using the Non-Moderated (maximum performance) mode.



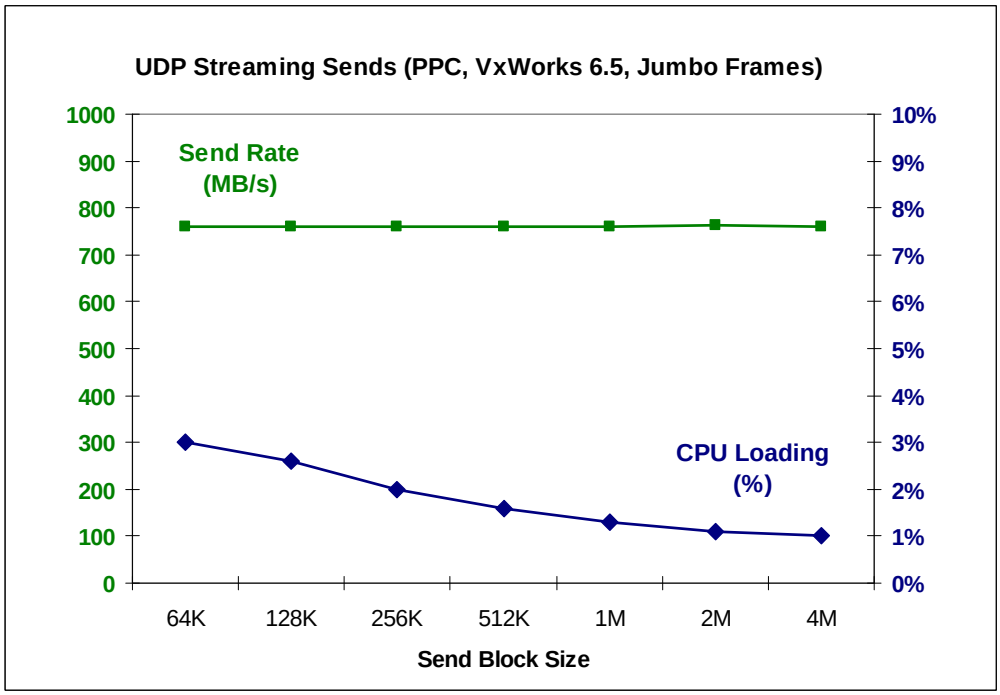


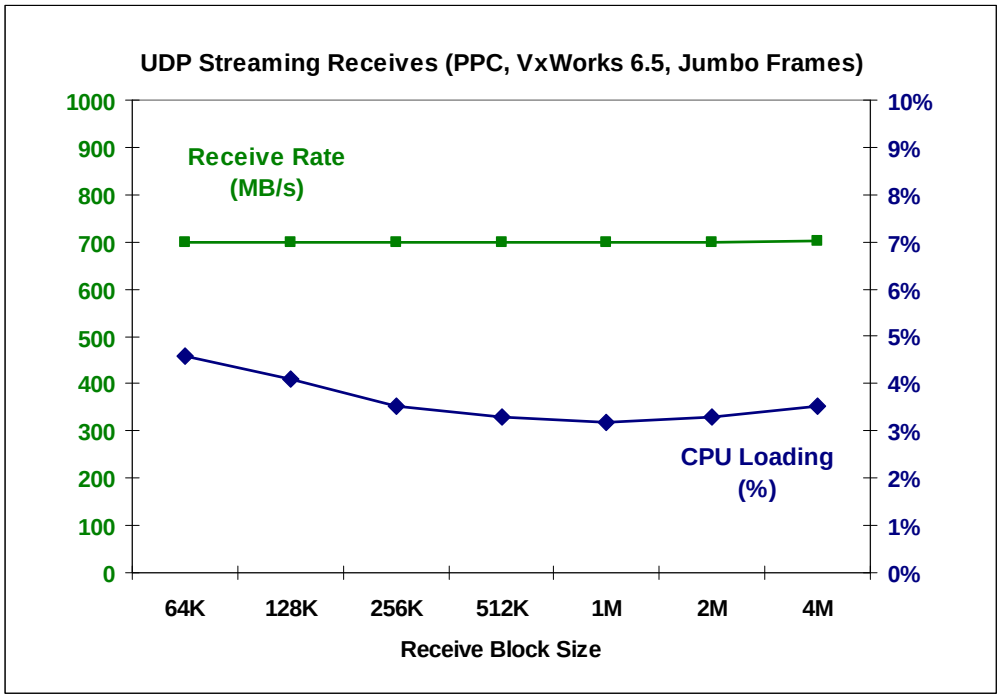

TCP Send Performance (Intel i7, VxWorks 6.8, Jumbo Frames)
Non-Moderated Mode
800
700
600
500
400
300
200
100
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Send Rate
(MB/s)
CPU Loading
(%)
1K
2K
4K
8K
16K
32K
64K
128K
256K
Send Block Size


TCP Receive Performance (Intel i7, VxWorks 6.8, Jumbo Frames)
Non-Moderated Mode
1000
900
800
700
600
500
400
300
200
100
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Receive Rate
(MB/s)
CPU Loading
(%)
1K
2K
4K
8K
16K
32K
64K
128K
256K
Receive Block Size

VxWorks Performance Results -- Streaming UDP API

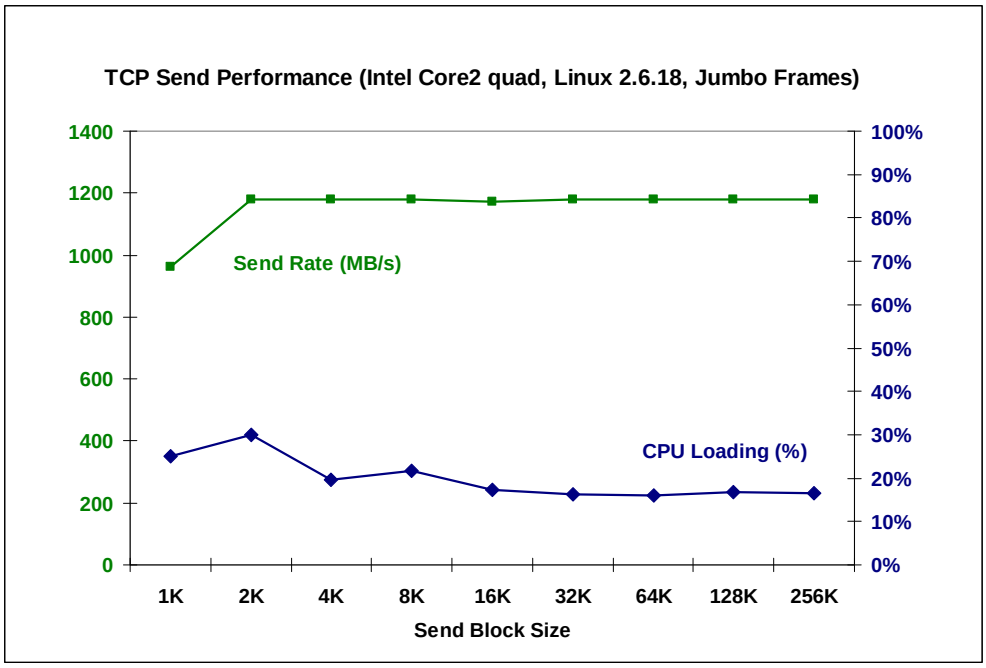
The following two charts show representative performance performing large block UDP sends or receives, accessing the XGE 10G through the Streaming UDP API. The charts show data rate in green (squares, upper lines), and CPU loading in blue (diamonds, lower lines), for a variety of send or receive block sizes. This data was taken using an embedded class processor base-board to host an XGE 10G XMC. The processor board used a PowerPC processor, running VxWorks 6.5. Standard 9K jumbo frames were used.

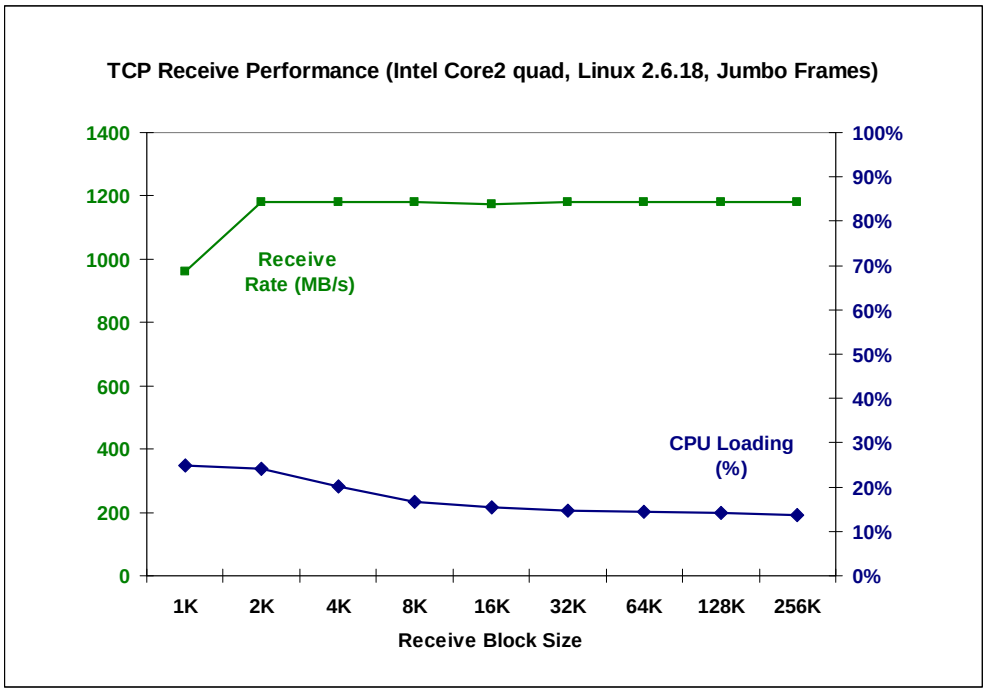




Linux Performance Results -- Standard Sockets API

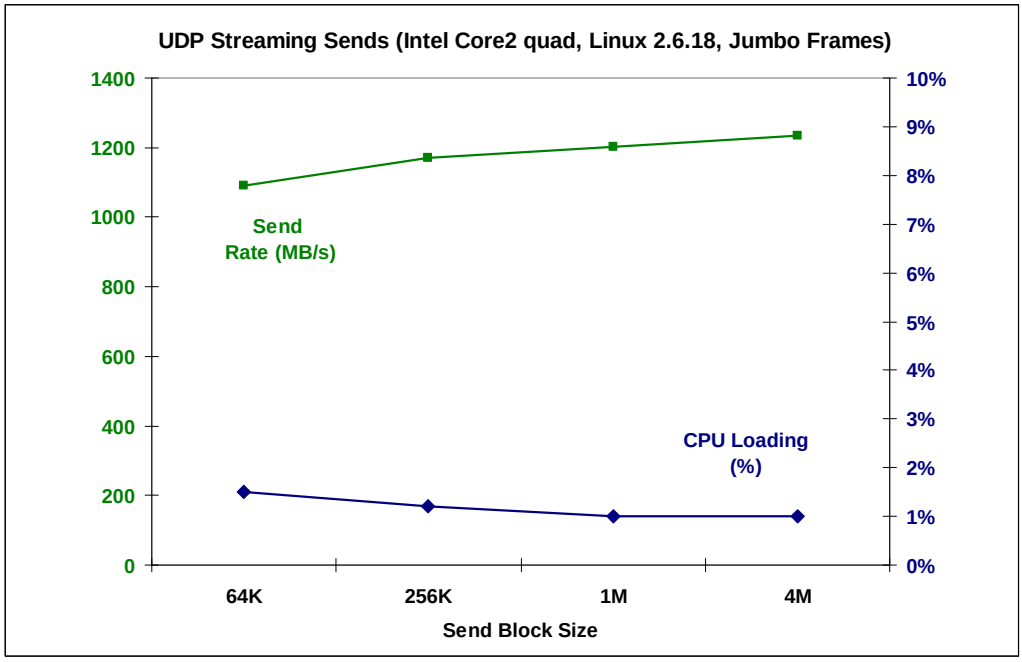
The following two charts show representative performance performing TCP sends or receives, accessing the XGE 10G through the standard Linux socket API. The charts show data rate in green (squares, upper lines), and CPU loading in blue (diamonds, lower lines), for a variety of send or receive block sizes. This data was taken using an embedded class processor base-board to host an XGE 10G XMC. The processor board used a quad-core Intel Core2 processor, running Linux 2.6.18. Standard 9K jumbo frames were used.

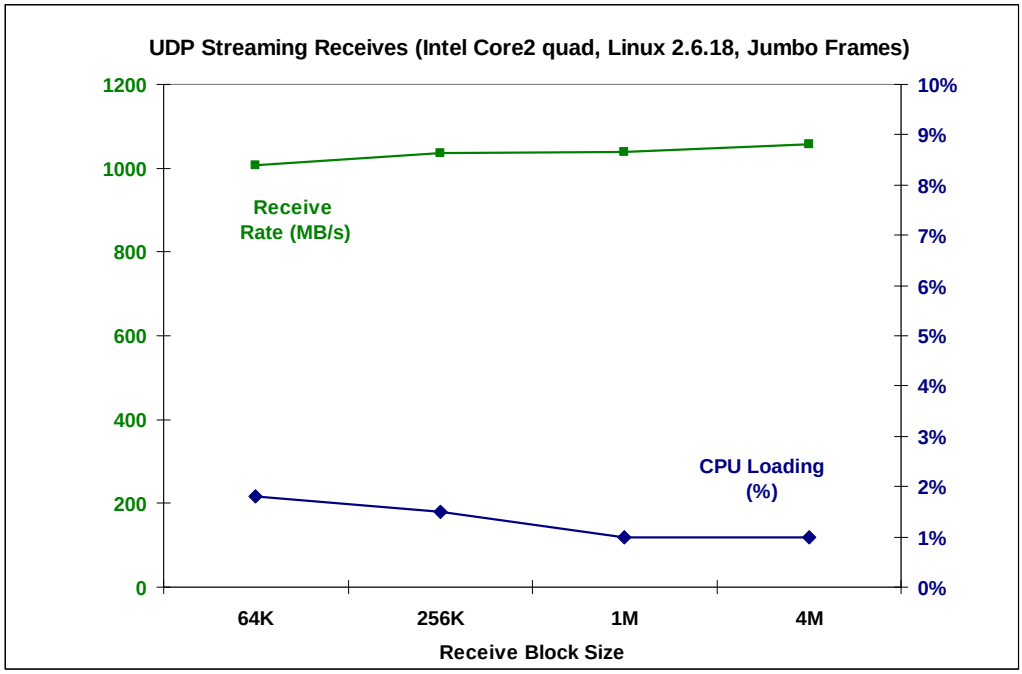




Linux Performance Results -- Streaming UDP API

The following three charts show representative performance performing large block UDP sends or receives, accessing the XGE 10G through the Streaming UDP API. The charts show data rate in green (squares, upper lines), and CPU loading in blue (diamonds, lower lines), for a variety of send or receive block sizes. This data was taken using an embedded class processor base-board to host an XGE 10G XMC. The processor board used a quad-core Intel Core2 processor, running Linux 2.6.18. Standard 9K jumbo frames were used.





Performance Results – Linux RDMA API

The two charts below show the bandwidth, latency and CPU utilization using the RDMA over Converged Enhanced Ethernet (RoCEE) protocol. Testing was performed between a host processor board with an Intel Core2 Quad (2.82GHz) processor (Client node) and another host processor Intel Xeon (2.40GHz) processor (Server node). A Fulcrum/Intel Monaco Data Center Bridged (DCB) switch was placed between the nodes and the link operated at 10G. Both nodes were running Linux Centos 6.2 (2.6.32) operating system.

Figure 1 shows the Round-Trip Time (RTT) Latency and corresponding client node CPU utilization. The RTT value is the time from sending a message of size n and receiving a response message of the same size divided by 2 (to account for the roundtrip). It is important to note that the driver provides 2 methods of receiving send and receive completions. One of these methods is to continuously poll a routine to check for completions – this method is very CPU intensive but can provide the lowest latency. The other method, which is used in for this testing, is to call a blocking function that returns when a completion is received from the hardware. This second method provides slightly higher latency but significantly lower CPU loading. Latency is measure from send start to receive complete, so for larger transfer sizes the latency is dominated by the actual transfer time.

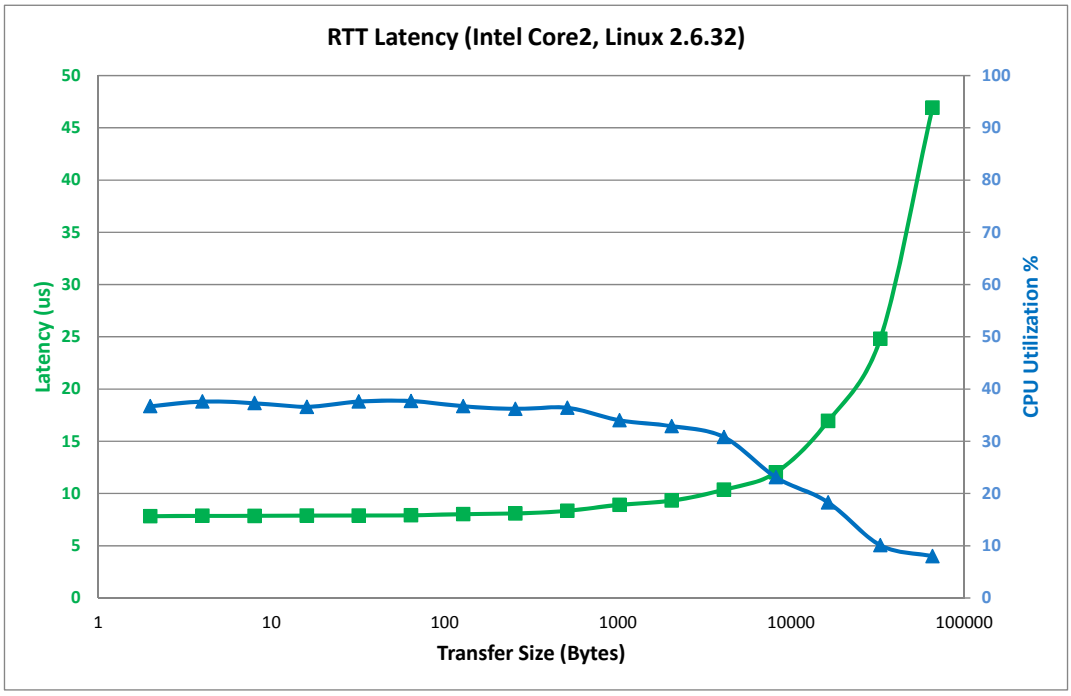


Figure 1: RDMA RTT Latency and CPU utilization

Figure 2 shows the RDMA write bandwidth and CPU utilization. Bandwidth hits the 10GE link maximum at transfer sizes of about 4KB. The CPU utilization starts out around 33% for 2 Byte transfers and declines to about 2% for 64KB transfers.

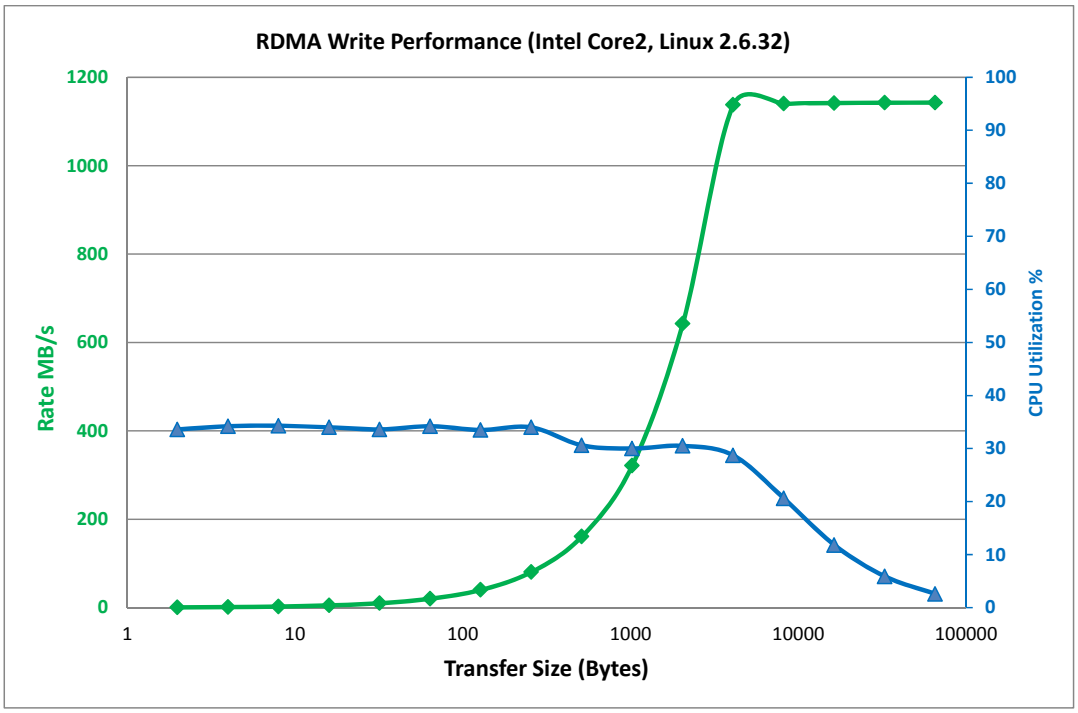


Figure 2: RDMA Write Bandwidth and CPU utilization.