



NVIDIA Accelerated IO (XLIO) Documentation Rev 3.31.2

Table of Contents

- Overview 11**
 - Document Revision History 11
- Release Notes..... 12**
 - System Requirements and Interoperability 12
 - System Requirements 12
 - XLIO Release Contents 14
 - Certified Applications 14
 - Changes and New Features 15
 - Changes and New Features in this Version 15
 - Bug Fixes in this Version 16
 - Known Issues 17
- Introduction to XLIO 31**
 - XLIO Overview 31
 - Basic Features 31
 - Target Applications 32
 - Advanced XLIO Features 32
- XLIO Library Architecture 34**
 - Top-Level 34
 - XLIO Internal Thread 34

Socket Types	35
XLIO Installation.....	36
Installing XLIO	36
Installing XLIO Binary as Part of NVIDIA Drivers	36
NVIDIA MLNX_OFED Driver for Linux	37
NVIDIA MLNX_EN Driver for Linux.....	37
Starting the Drivers.....	38
Running XLIO.....	38
Running Using Non-Root Permission	39
Libxlio-debug.so	40
XLIO Installation Options	41
Installing XLIO Binary Manually	41
Installing the XLIO Packages	41
Verifying XLIO Installation	44
Building XLIO From Sources	44
Verifying XLIO Compilation	44
Beta-Installing XLIO as DOCA profile	45
Uninstalling XLIO	45
Automatic XLIO Uninstallation	45
Manual XLIO Uninstallation.....	45
Upgrading libxlio.conf.....	46
Type Management / VPI Cards Configuration	46

Basic Performance Tuning	47
Binding XLIO to the Closest NUMA.....	47
Configuring the BIOS	48
XLIO Configuration.....	49
Configuring libxlio.conf	49
Configuring Target Application or Process.....	49
Configuring Socket Transport Control.....	50
Example of XLIO Configuration	52
XLIO Configuration Parameters	52
Beta Level Features Configuration Parameters	53
Loading XLIO Dynamically	54
Advanced Features	56
Direct Packet Control Plane (DPCP).....	56
TLS HW Offload	56
Prerequisites.....	56
Usage	56
Tuning of XLIO for TLS HW Offload in DNS-over-HTTPS (DoH) scenario	58
Tuning of XLIO for TLS HW Offload in Content Delivery Network (CDN) scenario.....	58
Supported Ciphers	58
Precision Time Protocol (PTP).....	59
Prerequisites.....	60


Usage	60
On-Device Memory	60
Prerequisites	61
Verifying On-Device Memory Capability in the Hardware	61
TCP_QUICKACK Threshold	62
XLIO Daemon Design	62
TAP Statistics	63
Getting Ring Protection Domain	64
NVME over TCP DIGEST Offload Tx (Alpha Level)	64
Prerequisites	64
Usage	64
SO_XLIO_ISOLATE option	65
Using sockperf with XLIO	66
XLIO Extra API	68
Overview of the XLIO Extra API	68
Using XLIO Extra API	69
Control Off-load Capabilities During Run-Time	69
Adding libxlio.conf Rules During Run-Time	69
Creating Sockets as Offloaded or Not-Offloaded	70
Zero Copy recvfrom()	70
Where:	71

Freeing Zero Copied Packet Buffers	71
SocketXtreme API	73
Usage	74
Verify SocketXtreme support.....	74
Replace the socket file descriptor with alternative identifier.....	75
Get ring file descriptors(s) from socket file descriptor	75
Poll ring fd	76
Processing the completions	77
Freeing packets and buffers.....	78
Packet Filter Callback API	78
Dump fd Statistics using XLIO Logger	79
"Dummy Send" to Improve Low Message Rate Latency.....	80
Verifying "Dummy Send" Capability in HW	81
"Dummy Packets" Statistics.....	81
Control the Library	82
CMSG_XLIO_IOCTL_USER_ALLOC	84
Usage Example	84
XLIO Socket API.....	87
General Information	87
Global Initialization	88
XLIO Polling Groups.....	90
XLIO Sockets	93
XLIO Socket Events	97

TX Data Path	99
TX Data Path - Zerocopy Completions	102
RX Data Path	103
Monitoring, Debugging, and Troubleshooting	106
Monitoring - the xlio_stats Utility	106
Examples	108
Example 1	108
Example 2	110
Example 3	111
Example 4	114
Example 5	114
Example 6	115
Example 7	116
Example 8	118
Memory Report	119
Debugging	120
XLIO Logs	120
Ethernet Counters	121
tcpdump	121
NIC Counters	121
Peer Notification Service	121
Troubleshooting	121
Appendixes	126

Appendix: Sockperf - UDP/TCP Latency and Throughput Benchmarking Tool	126
Overview	126
Advanced Statistics and Analysis	127
Configuring the Routing Table for Multicast Tests.....	128
Latency with Ping-pong Test	128
UDP Ping-pong	128
TCP Ping-pong	129
TCP Ping-pong using XLIO.....	129
Bandwidth and Packet Rate with Throughput Test	130
UDP MC Throughput	130
UDP MC Throughput using XLIO	131
UDP MC Throughput Summary	132
sockperf Subcommands	132
Additional Options	133
Sending Bursts	136
Debugging sockperf.....	137
Troubleshooting sockperf	137
Appendix: Nginx.....	137
Appendix: Multicast Routing	138
Common Abbreviations, Typography and Related Documents.....	139
Glossary	139
Typography.....	140
Related Documentation	141

User Manual Revision History	142
Release Notes Revision History	144
Release Notes Change History	144
Bug Fixes History.....	147

 You can download a pdf [here](#).

Overview

The NVIDIA® Accelerated IO (XLIO) SW library boosts the performance of TCP/IP applications based on NGINX (CDN, DoH Etc.) and storage solutions as part of the SPDK.

Coupling XLIO with the acceleration capabilities of NVIDIA ConnectX®-6 Dx, NVIDIA ConnectX®-7, or NVIDIA Bluefield®-3 data processing unit (DPU) provides breakthrough performance of Transport Layer Security (TLS) encryption/decryption. It also enables working with features such as HW LRO/TSO and Striding-RQ which increase TCP performance, without application code changes and using a standard socket API.

XLIO is an open-source project - see its repository [here](#).

Further information on this product can be found in the following MLNX_OFED documents:

- [Release Notes](#)
- [User Manual](#)

Document Revision History

For the list of changes made to this document, refer to [User Manual Revision History](#).

Release Notes

Revision	Date	Description
3.31.2	August 14, 2024	Initial release of this document.

The release note pages provide information on the NVIDIA Accelerated IO (XLIO), such as changes and new features, system requirements, interoperability parameters, reports on software known issues, and bug fixes.

- [System Requirements and Interoperability](#)
- [Changes and New Features](#)
- [Bug Fixes in this Version](#)
- [Known Issues](#)

System Requirements and Interoperability

System Requirements

The following table presents the currently certified combinations of stacks and platforms and supported CPU architectures for the current XLIO version.

Specification	Value
Network Adapter Cards	NVIDIA Bluefield-3, 200 GbE, Dual Port, 32GB RAM
	NVIDIA ConnectX-7, 200 GbE, Dual Port
	NVIDIA ConnectX-6 Dx, 100GbE, Dual Port
Firmware	BlueField-3 32.42.1000

Specification	Value
	BlueField-2 24.42.1000
	ConnectX-7 28.42.1000
	ConnectX-6 Dx 22.42.1000
Driver Stack	<ul style="list-style-type: none"> • MLNX_OFED v24.07-0.6.1.0 • MLNX_EN v24.07-0.6.1.0
Supported Operating Systems and Kernels	<ul style="list-style-type: none"> • Ubuntu 22.04 • RH8.6 and Oracle Linux 8.x UEK7
CPU Architecture	x86_64, NVIDIA BlueField-3 Arm
Minimum memory requirements	1 GB of free memory for installation 800 MB per process running with XLIO
Minimum disk space requirements	1 GB
Transport	Ethernet

XLIO Release Contents

Content	Description
Binary RPM and DEB packages for 64-bit architecture for Linux distribution	libxlio-X.XX-X.x86_64.rpm libxlio-devel-X.XX-X.x86_64.rpm libxlio-utils-X.XX-X.x86_64.rpm libxlio-dev_X.XX-X.XXXXXXX_amd64.deb libxlio-utils_X.XX-X.XXXXXXX_amd64.deb libxlio_X.XX-X.XXXXXXX_amd64.deb
Documentation	XLIO Release Notes XLIO User Manual

Certified Applications

The XLIO library has been successfully tested and is certified to work with the applications listed in the following table.


Application	Company / Source	Type	Notes
sockperf	NVIDIA (Open Source)	Bandwidth and Latency Benchmarking	Version 3.10 (https://github.com/mellanox/sockperf)

Changes and New Features


Changes and New Features in this Version

Feature/Category	Description
Allocation Logic Parameters	Changed the default value of the allocation logic parameters <code>XLIO_RING_ALLOCATION_LOGIC_TX</code> and <code>XLIO_RING_ALLOCATION_LOGIC_RX</code> to 20 (Per-Thread).
Socket API	Added XLIO Socket API as an extra API - see XLIO Socket API section.
Bug Fixes	See Bug Fixes section.

Important Notes

 As of XLIO v3.10, DEBUG-level logs will not be available in the release builds. Alternatively, you can use [libxlio-debug.so](#) to obtain those logs. For details on how to use [libxlio-debug.so](#), please refer to [Libxlio-debug.so](#) section.

 **libnl1** is no longer supported. Please use **libnl3** instead.

 Bonding Active-Backup is not supported. RoCE LAG is a feature meant for mimicking Ethernet bonding for IB devices and is available for dual-port cards only. XLIO cannot offload traffic in cases where RoCE LAG is enabled too. In RoCE LAG mode, instead of having an IB device per physical port (for example, `mlx5_0` and `mlx5_1`), only one IB device is present for both ports.

⚠ Direct Packet Control Plane (DPCP) provides a unified, flexible interface for programming NVIDIA NICs and comes as part of OFED. The DPCP version must be v1.1.44 and above. For further details, please see [Direct Packet Control Plane \(DPCP\)](#).

- ⚠** TLS Rx offload related notes:
- TLS Rx offload supports up to 64K concurrent connections
 - TLS Rx offload for IPv6 is not supported as it may cause unexpected behavior
 - TLS Rx offload requires OpenSSL 3.0.2 or higher and kTLS support from the kernel

Bug Fixes in this Version

The table below lists the bugs that have been fixed in the current XLIO version.

Internal Ref. Number	Details
3858121	Description: Fixed a race condition that occurred when XLIO was unloaded, which could lead to data corruption in STAT_FILE due to concurrent calls to the stats destructors. Note: Live stats were not affected by this issue.
	Keywords: Data corruption; STAT_FILE; destructor
	Discovered in Version: 3.30.5
	Fixed in Version: 3.31.2
3854806	Description: Fixed the issue where XLIO Ring might not have returned buffers to the global pool, which could lead to higher TX buffer consumption.
	Keywords: Tx buffers global pool
	Discovered in Version: 3.30.5
	Fixed in Version: 3.31.2

Internal Ref. Number	Details
3784248	Description: Fixed the issue where using the same port number for both IPV4 and IPV6 on an Nginx UDP listen socket with the "reuseport" directive did not work.
	Keywords: reuseport; Nginx; UDP; IPV6
	Discovered in Version: 3.30.5
	Fixed in Version: 3.31.2
3598943	Description: Fixed the issue where using CQ adaptive moderation could have led to a very high interrupt rate.
	Keywords: CQ adaptive moderation; high interrupt rate
	Discovered in Version: 3.30.5
	Fixed in Version: 3.31.2

Known Issues

The following is a list of general existing limitations and known issues of the various components of XLIO.

 Since XLIO has been inherited from Messaging Accelerator (VMA) v9.2.2, some issues point to [\(VMA\)](#).

Internal Ref. Number	Details
3795997	Description: When the aggregated data to be sent exceeds the TCP send window, XLIO will postpone the transmission (Tx operation) until all previously sent data is acknowledged by the remote side.

Internal Ref. Number	Details
	<p>Workaround: Increase the TCP receive window size on the remote side.</p> <p>Keywords: TSO; quick ack; delayed ack; TCP window</p> <p>Discovered in Version: 3.31.2</p>
3795997	<p>Description: When aggregated data to send is bigger than the TCP send window allows - XLIO will postpone TX operation unless all previously sent data is acked by the remote side.</p> <p>Workaround: Increase TCP receive window size on the remote side.</p> <p>Keywords: TSO, Quick ack, Delayed ack, TCP window</p> <p>Discovered in Version: 3.30.5</p>
3865579	<p>Description: For small packets, up to 205 bytes, the removal of the BlueFlame feature from XLIO causes latency degradation. Expect up to a 400 ns increase in latency.</p> <p>Workaround: N/A</p> <p>Keywords: latency, blue flame</p> <p>Discovered in Version: 3.30.5</p>

Internal Ref. Number	Details
3654779	<p>Description: In the current state of the library, the TCP keep-alive feature is partially supported. The known limitations are:</p> <ol style="list-style-type: none"> The keep-alive internal parameters are as follows: <ol style="list-style-type: none"> TCP_KEEPINTVL - 75,000ms or 75 seconds (1:15 minutes) TCP_KEEPCNT - 9 (probes to send before timeout) The following setsockopt options are unsupported: <ol style="list-style-type: none"> setsockopt(sock, IPPROTO_TCP, TCP_KEEPINTVL, &interval, sizeof(interval)); - Change the intervals between the keep-alive probes not operational setsockopt(sock, IPPROTO_TCP, TCP_KEEPCNT, &retries, sizeof(retries)); - Change the number of probes not operational The following system configurations are disregarded: <pre> `sysctl -w net.ipv4.tcp_keepalive_intvl=60` `sysctl -w net.ipv4.tcp_keepalive_probes=3` </pre> The contents of the following files are disregarded: <pre> /proc/sys/net/ipv4/tcp_keepalive_intvl /proc/sys/net/ipv4/tcp_keepalive_probes </pre> <p>Workaround: N/A</p> <p>Keywords: keep-alive, TCP_KEEPALIVE</p> <p>Discovered in Version: 3.10</p>
3440429	<p>Description: XLIO fails to create RFS rule and prints the following error:</p> <pre> Create RFS flow failed, Tag: Y, Flow: dst: X.X.X.X:X, src: X.X.X.X:X, proto: TCP, family: INET, Priority: 1, errno: 17 - File exists </pre> <p>This happens when a new socket allocates the same fd as a socket in cleanup process.</p> <p>Workaround: Set XLIO_TCP_ABORT_ON_CLOSE=1 to lower the chances of such a scenario to take place.</p> <p>Keywords: RFS; steering failure error</p> <p>Discovered in Version: 3.0.2</p>

Internal Ref. Number	Details
2970828	<p>Description: XLIO ignores all signals except SIGINT signal, occasionally resulting in blocked IO calls.</p> <p>Workaround: N/A</p> <p>Keywords: Signal; SIGINT; IO</p> <p>Discovered in Version: 2.1.4</p>
3204491	<p>Description: Note the following limitations when working with setsockopt for UDP multicast socket.</p> <ol style="list-style-type: none"> 1. Setting either IP_DROP_MEMBERSHIP or IP_DROP_SOURCE_MEMBERSHIP reverts IP_MULTICAST_LOOP=0 to 1. 2. Only the first multicast group will be offloaded by XLIO using IP_ADD_SOURCE_MEMBERSHIP or MCAST_JOIN_SOURCE_GROUP. The rest of the groups will be handled by the Kernel. 3. Binding to a specific IP (not any_address) after setting multicast setsockopt options will result in binding to any_address instead. <p>Workaround: N/A</p> <p>Keywords: multicast; setsockopt; bind</p> <p>Discovered in Version: 2.1.4</p>
3217617	<p>Description: When XLIO binds to any IP, resource allocation is required for each IP per port. This action may take up to several seconds with a limit of 64K steerings per family.</p> <p>Workaround: To avoid this scenario, bind a socket to a specific IP instead.</p> <p>Keywords: Bind to any IP; resource allocation</p> <p>Discovered in Version: 2.0.6</p>
3190147	<p>Description: TCP User Timeout options are not supported with select(); pselect(); poll(); ppoll(); __poll_chk(); and __ppoll_chk.</p>

Internal Ref. Number	Details
	<p>Workaround: Application may use epoll instead.</p> <p>Keywords: TCP; user timeout; select; poll; ppoll</p> <p>Discovered in Version: 2.0.6</p>
3217629	<p>Description: iperf3 usage of longjmp operation in the signal handler results in abandoning previous thread context, leaving the state of internal objects in an undefined state.</p> <p>Workaround: N/A</p> <p>Keywords: iperf3; signal handler termination</p> <p>Discovered in Version: 2.0.6</p>
3231754/3208316	<p>Description: Traffic over IPv6 link-local addresses is not supported.</p> <p>Workaround: N/A</p> <p>Keywords: IPv6; link-local</p> <p>Discovered in Version: 2.0.6</p>
3231768	<p>Description: IPv6 Flow Labels are not supported.</p> <p>Workaround: N/A</p> <p>Keywords: IPv6 flow labels</p> <p>Discovered in Version: 2.0.6</p>

Internal Ref. Number	Details
3231781	<p>Description: The following IPv6 multicast socket options are not supported:</p> <ul style="list-style-type: none"> • IPV6_MULTICAST_LOOP • MCAST_BLOCK_SOURCE • MCAST_UNBLOCK_SOURCE <p>Workaround: N/A</p> <p>Keywords: IPV6_MULTICAST_LOOP; MCAST_BLOCK_SOURCE; MCAST_UNBLOCK_SOURCE; socket; IPv6</p> <p>Discovered in Version: 2.0.6</p>
3232106	<p>Description: SO_MAX_PACING_RATE socket option is not supported.</p> <p>Workaround: N/A</p> <p>Keywords: SO_MAX_PACING_RATE; socket</p> <p>Discovered in Version: 2.0.6</p>
3136901	<p>Description: If a multi-threaded application calls exit() syscall while other threads call socket API concurrently, this can lead to a race between XLIO library destructor and intercepted socket API or in rare cases to a segmentation fault.</p> <p>Workaround: It is recommended to finish running threads before process termination. Alternatively, avoid calling socket API concurrently with exit() syscall.</p> <p>Keywords: multithreaded applications</p> <p>Discovered in Version: 1.3.5</p>
2667588	<p>Description: XLIO conf file does not support IPv6 (see XLIO Configuration Parameters).</p>

Internal Ref. Number	Details
	<p>Workaround: N/A</p> <p>Keywords: xlio.conf; IPv6</p> <p>Discovered in Version: 1.2.9</p>
N/A	<p>Description: Socket API breaks for closed UDP sockets when using XLIO_NGINX_UDP_POOL_SIZE != 0, as the socket is not entirely closed. Socket operations will not fail after the socket is closed.</p> <p>Workaround: N/A</p> <p>Keywords: UDP; pool</p> <p>Discovered in Version: 1.2.9</p>
1542628	<p>Description: Device memory programming is not supported on VMs that lack Blue Flame support.</p> <p>Workaround: N/A</p> <p>Keywords: MEMIC, Device Memory, Virtual Machine, Blue flame</p> <p>Discovered in Version: 1.0.6</p>
–	<p>Description: If XLIO runs when XLIO_HANDLE_SIGINTR is enabled, an error message might be written upon exiting.</p> <p>Workaround: Ignore the error message, or run XLIO with XLIO_HANDLE_SIGINTR disabled.</p> <p>Keywords: XLIO_HANDLE_SIGINTR, error message</p> <p>Discovered in Version: 1.0.6</p>

Internal Ref. Number	Details
965237	<p>Description: The following sockets APIs are directed to the OS and are not offloaded by XLIO:</p> <ul style="list-style-type: none"> • int socketpair(int domain, int type, int protocol, int sv[2]); • int dup(int oldfd); • int dup2(int oldfd, int newfd); <p>Workaround: N/A</p> <p>Keywords: sockets, socketpair, dup, dup2</p>
965227	<p>Description: Multicast (MC) loopback within a process is not supported by XLIO:</p> <ul style="list-style-type: none"> • If an application process opens 2 (or more) sockets on the same MC group they will not get each other's traffic. <ul style="list-style-type: none"> Note: MC loopback between different XLIO processes always work. • Both sockets will receive all ingress traffic coming from the wire <p>Workaround: N/A</p> <p>Keywords: Multicast, Loopback</p>
965227	<p>Description: MC loopback between XLIO and the OS limitation.</p> <ul style="list-style-type: none"> • The OS will reject loopback traffic coming from the NIC • MC traffic from the OS to XLIO is functional <p>Workaround: N/A</p> <p>Keywords: Multicast, Loopback</p>
965227	<p>Description: MC loopback Tx is currently disabled and setsockopt (IP_MULTICAST_LOOP) is not supported.</p>

Internal Ref. Number	Details
	<p>Workaround: N/A</p> <p>Keywords: Multicast, Loopback</p>
1011005	<p>Description: XLIO SELECT option supports up to 200 sockets in TCP.</p> <p>Workaround: Use ePoll that supports up to 6000 sockets.</p> <p>Keywords: SockPerf</p>
977899	<p>Description: An unsuccessful trial to connect to a local interface, is reported by XLIO as Connection timeout rather than Connection refused.</p> <p>Workaround: N/A</p> <p>Keywords: Verification</p>
1019085	<p>Description: Poll is limited with the number of sockets.</p> <p>Workaround: Use ePoll for large number of sockets (tested up to 6000)</p> <p>Keywords: Poll, ePoll</p>
–	<p>Description: The following XLIO_PANIC will be displayed when there are not enough open files defined on the server:</p> <pre data-bbox="414 1077 1568 1109">XLIO PANIC : si[fd=1023]:51:sockinfo() failed to create internal epoll (ret=-1 Too many open files</pre> <p>Workaround: Verify that the number of max open FDs (File Descriptors) in the system (ulimit -n) is twice as number of needed sockets. XLIO internal logic requires one additional FD per offloaded socket.</p> <p>Keywords: XLIO_PANIC while opening large number of sockets</p>

Internal Ref. Number	Details
–	<p>Description: When a XLIO -enabled application is running, there are several cases when it does not exit as expected pressing CTRL-C.</p> <p>Workaround: Enable SIGINT handling in XLIO, by using:</p> <pre>#export XLIO_HANDLE_SIGINTR=1</pre> <p>Keywords: The XLIO application does not exit when you press CTRL-C.</p>
–	<p>Description: XLIO does not support network interface changes during runtime</p> <p>Workaround: N/A</p> <p>Keywords: Dynamic network interface changes</p>
–	<p>Description: XLIO behavior of epoll EPOLLET (Edge Triggered) and EPOLLOUT flags with TCP sockets differs between OS and XLIO.</p> <ul style="list-style-type: none"> • XLIO - triggers EPOLLOUT event every received ACK (only data, not syn/fin) • OS - triggers EPOLLOUT event only after buffer was full <p>Workaround: N/A</p> <p>Keywords: XLIO behavior of epoll EPOLLET (Edge Triggered) and EPOLLOUT flags with TCP sockets</p>
–	<p>Description: XLIO does not close connections located on the same node (sends FIN to peers) when its own process is terminated.</p> <p>Workaround: N/A</p> <p>Keywords: XLIO connection</p>
–	<p>Description: XLIO is not closed (sends FIN to peers) when its own process is terminated when the /etc/init.d/XLIO is stopped.</p> <p>Workaround: Launch /etc/init.d/XLIO start</p>

Internal Ref. Number	Details
	<p>Keywords: XLIO connection</p>
–	<p>Description: When a non-offloaded process joins the same MC address as another XLIO process on the same machine, the non-offloaded process does not get any traffic.</p> <p>Workaround: Run both processes with XLIO</p> <p>Keywords: MC traffic with XLIO process and non XLIO process on the same machine</p>
–	<p>Description: Occasionally, epoll with EPOLLONESHOT does not function properly.</p> <p>Workaround: N/A</p> <p>Keywords: epoll with EPOLLONESHOT</p>
–	<p>Description: Occasionally, when running UDP SFNT-STREAM client with poll muxer flag, the client-side ends with an expected error:</p> <pre>ERROR: Sync messages at end of test lost ERROR: Test failed.</pre> <p>This only occurs with poll flag</p> <p>Workaround: Set a higher acknowledgment waiting time value in the sfnt-stream.</p> <p>Keywords: SFNT-STREAM UDP with poll muxer flag ends with an error on client-side</p>
–	<p>Description: Occasionally, SFNT-STREAM UDP client hangs when running multiple times.</p> <p>Workaround: Set a higher acknowledgment waiting time value in the sfnt-stream.</p> <p>Keywords: SFNT-STREAM UDP client hanging issue</p>

Internal Ref. Number	Details
–	<p>Description: Ethernet loopback functions only if both sides are either off-loaded or not-offloaded.</p> <p>Workaround: N/A</p> <p>Keywords: Ethernet loopback is not functional between the XLIO and the OS</p>
–	<p>Description: The following error may occur when running netperf TCP tests with XLIO:</p> <pre data-bbox="425 558 1041 590">remote error 107 'Transport endpoint is not connected'</pre> <p>Workaround: Use netperf 2.6.0</p> <p>Keywords: Error when running netperf 2.4.4 with XLIO</p>
–	<p>Description: Occasionally, a packet is not sent if the socket is closed immediately after send() (also for blocking socket)</p> <p>Workaround: Wait several seconds after send() before closing the socket</p> <p>Keywords: A packet is not sent if the socket is closed immediately after send()</p>
–	<p>Description: It can take for XLIO more time than the OS to return from an iomux call if all sockets in this iomux are empty sockets</p> <p>Workaround: N/A</p> <p>Keywords: iomux call with empty sockets</p>
–	<p>Description: Sharing of HW resources between the different working threads might cause lock contentions which can affect performance.</p> <p>Workaround: Use different ring allocation logic.</p> <p>Keywords: Issues with performance with some multi-threaded applications</p>

Internal Ref. Number	Details
–	<p>Description: XLIO does not support broadcast traffic.</p> <p>Workaround: Use libxlio.conf to pass broadcast through OS</p> <p>Keywords: No support for direct broadcast</p>
–	<p>Description: Directing XLIO to access non-valid memory area will cause a segmentation fault.</p> <p>Workaround: N/A</p> <p>Keywords: There is no non-valid pointer handling in XLIO</p>
–	<p>Description: XLIO allocates resources on the first connect/send operation, which might take up to several tens of milliseconds.</p> <p>Workaround: N/A</p> <p>Keywords: First connect/send operation might take more time than expected</p>
–	<p>Description: Calling select upon shutdown of socket will return “ready to write” instead of timeout.</p> <p>Workaround: N/A</p> <p>Keywords: Calling select() after shutdown (write) returns socket ready to write, while select() is expected to return timeout</p>
–	<p>Description: XLIO does not raise sigpipe in connection shutdown.</p> <p>Workaround: N/A</p> <p>Keywords: XLIO does not raise sigpipe</p>

Internal Ref. Number	Details
–	<p>Description: XLIO polls the CQ for packets; if no packets are available in the socket layer, it takes longer.</p> <p>Workaround: N/A</p> <p>Keywords: When there are no packets in the socket, it takes longer to return from the read call</p>
–	<p>Description: Select with more than 1024 sockets is not supported</p> <p>Workaround: Compile XLIO with SELECT_BIG_SETSIZE defined.</p> <p>Keywords: 1024 sockets</p>
–	<p>Description: Possible performance degradation running NGINX QUIC.</p> <p>Workaround: Use XLIO_CQ_MODERATION_ENABLE=0.</p> <p>Keywords: QUIC, NGINX, XLIO_CQ_MODERATION_ENABLE</p>

Introduction to XLIO

XLIO Overview

The Accelerated IO (XLIO) library is a network-traffic offload, dynamically-linked user-space Linux library that transparently enhances the performance of socket-based networking-heavy applications over an Ethernet network. In addition, XLIO exposes standard socket APIs with kernel-bypass architecture, enabling a hardware-based direct copy between an application's user-space memory and the network interface.

The XLIO library accelerates TCP and UDP socket applications by offloading traffic from the user-space directly to the network interface card (NIC) without going through the kernel and the standard IP stack (kernel-bypass). XLIO increases overall traffic packet rate and improves latency and CPU utilization for NGINX and SPDK based applications.

Coupling XLIO with NVIDIA ConnectX-6 Dx, NVIDIA ConnectX-7 or NVIDIA BlueField-3 data processing unit (DPU) acceleration capabilities provides breakthrough performance of Transport Layer Security (TLS) encryption, without application code changes, using a standard socket API.

Basic Features

The XLIO library utilizes the direct hardware access and advanced polling techniques of NVIDIA network cards. Utilization of Ethernet's direct hardware access enables the XLIO kernel bypass, which causes the XLIO library to bypass the kernel's network stack for all IP network traffic transmit and receive socket API calls. Thus, applications using the XLIO library gain many benefits, including:

- Reduced context switches and interrupts, which result in:
 - Higher throughput
 - Requests per second
 - Number of new connections per second
 - Improved CPU utilization
 - Lower latency
- Minimal buffer copies between user data and hardware - XLIO needs a single copy to transfer a unicast or multicast offloaded packet between hardware and the application's data buffers.

Target Applications


Good application candidates for XLIO include, but are not limited to:

- Applications that require NGINX TCP/IP performance acceleration for HTTP/HTTPS traffic, DoH, load-balancing, reverse proxy, and CDN. In addition, hardware offload capabilities for TLS Encryption.
- Applications that require accelerated SPDK (Storage Performance Development Kit) performance over NVME-over-TCP.

Advanced XLIO Features

The XLIO library provides several significant advantages:

- The underlying wire protocol used for the unicast and multicast solution is standard TCP and UDP IPv4/IPv6, which is interoperable with any TCP/UDP/IP networking stack. Thus, the opposite side of the communication can be any machine with any OS and can be located on an Ethernet network.

 XLIO uses a standard protocol that enables an application to use the XLIO for asymmetric acceleration purposes. A “TCP server-side” application, a “multicast consuming” only, or “multicast publishing” only application can leverage this while remaining compatible with Ethernet peers.

- Kernel bypass for unicast and multicast transmit and receive operations. This delivers much lower CPU overhead since TCP/IP stack overhead is not incurred
- Better CPU utilization by eliminating the context switch costs using kernel bypass
- Reduced number of context switches. All XLIO software is implemented in user space in the user application’s context. This allows the server to process a significantly higher packet rate than would otherwise be possible
- No buffer copies in the kernel
- Fewer hardware interrupts for received/transmitted packets
- Fewer queue congestion problems witnessed in standard TCP/IP applications
- Supports legacy socket applications - no need for application code rewrite
- Maximizes messages per second (MPS) rates
- Minimizes message latency
- Reduces latency spikes (outliers)

- Lowers the CPU usage required to handle traffic
- Hardware offload of TLS encryption for Tx and Rx paths
- Zero-copy extra API for TX and RX

XLIO Library Architecture

Top-Level

The XLIO library is a dynamically linked user-space library. The use of the XLIO library does not require any code changes or recompiling of user applications. Instead, it is dynamically loaded via the Linux OS environment variable, `LD_PRELOAD`. However, it is possible to load the XLIO library dynamically without using the `LD_PRELOAD` parameter, which requires minor application modifications.

When a user application transmits TCP and UDP, unicast and multicast IPv4/IPv6 data, or listens for such network traffic data, the XLIO library:

- Intercepts the socket Receive and Send calls made to the stream socket or datagram socket address families.
- Implements the underlying work in user space (instead of allowing the buffers to pass on to the usual OS network stack).

XLIO implements native Socket API. The implementation utilizes NVIDIA Ethernet NICs, enabling the packets to pass directly between the user application and the Ethernet NIC, bypassing the kernel and its TCP/UDP handling network stack.

The application can implement the code in native Socket API, without making any changes. The XLIO library does all the heavy lifting under the hood, while transparently presenting the same standard socket API to the application, thus redirecting the data flow.

The XLIO library operates in a standard networking stack fashion to serve multiple network interfaces.

The XLIO library behaves according to the way the application calls the *bind*, *connect*, and *setsockopt* directives and the administrator sets the route lookup to determine the interface to be used for the socket traffic. The library knows whether data is passing to or from an Ethernet NIC. If the data is passing to/from a supported Ethernet NIC, the XLIO library intercepts the call and does the bypass work. If the data is passing to/from an unsupported Ethernet NIC, the XLIO library passes the call to the usual kernel libraries responsible for handling network traffic. Thus, the same application can listen in on multiple Ethernet NICs, without requiring any configuration changes for the hybrid environment.

XLIO Internal Thread

The XLIO library has an internal thread that is responsible for performing general operations in order to maintain a high level of performance. These operations are performed in the context of a separate thread to that of the main application.

The main activities performed by the internal thread are:

- Poll the CQ if the application does not do so to avoid packet drops
- Synchronize the card clock with the system clock
- Handle any application housekeeping TCP connections (which should not impact its data path performance). For example: sending acknowledgments, retransmissions etc...
- Handle the final closing of TCP sockets
- Update XLIO statistics tool data
- Update epoll file descriptor contexts for available non-offloaded data
- Handle bond management

There are several parameters by which the user can set the characteristics of the internal thread. See section [XLIO Configuration Parameters](#) for a detailed description.

Socket Types

The following Internet socket types are supported:

- Datagram sockets, also known as connectionless sockets, use User Datagram Protocol (UDP)
- Stream sockets, also known as connection-oriented sockets, use Transmission Control Protocol (TCP)

XLIO Installation

- [Installing XLIO](#)
- [Running XLIO](#)
- [XLIO Installation Options](#)
- [Uninstalling XLIO](#)
- [Upgrading libxlio.conf](#)
- [Type Management / VPI Cards Configuration](#)
- [Basic Performance Tuning](#)

Installing XLIO

Before you begin, verify you are using a supported operating system and a supported CPU architecture for your operating system. See supported combinations listed in [System Requirements and Interoperability v1.1.8](#).

The current XLIO version can work on top of both MLNX_OFED driver stack that supports Ethernet and on a lighter driver stack, MLNX_EN that supports only Ethernet.

The XLIO library is delivered as a user-space library, and is called libxlio.so.X.Y.Z.

XLIO can be installed using one of the following methods:

- As part of NVIDIA drivers (described on this page)
- Manually (see [Installing the XLIO Packages](#))
- Beta - As part of DOCA (see [Installing XLIO as DOCA profile](#))

Installing XLIO Binary as Part of NVIDIA Drivers

XLIO is integrated into the NVIDIA drivers (MLNX_OFED/MLNX_EN), therefore, it is recommended to install XLIO automatically when installing the NVIDIA drivers as it depends on drivers' latest firmware, libraries, and kernel modules. This installation assures XLIO's correct functionality. Since the drivers have a plethora of distributions for RHEL, Ubuntu and others, you will have to correctly select the drivers' version that matches your distribution.

This option suits users who want to install a new XLIO version or upgrade to the latest XLIO version by overriding the previous one.

NVIDIA MLNX_OFED Driver for Linux

1. Download the latest MLNX_OFED driver from [here](#).
2. Install the XLIO packages.

```
./mlnxofedinstall --xlio
```

3. Verify the installation was completed successfully.

```
/etc/infiniband/info
```

NVIDIA MLNX_EN Driver for Linux

1. Download the latest MLNX_EN driver from [here](#).
2. Install the XLIO packages.

```
./install --xlio
```

3. Verify the installation was completed successfully.

```
$ cat /etc/infiniband/info
#!/bin/bash

echo prefix=/usr
echo Kernel=4.18.0-240.el8.x86_64
echo
echo "Configure options: --with-core-mod
--with-user_mad-mod --with-user_access-mod --with-addr_trans-mod
--with-mlx54-mod --with-mlx54_en-mod --with-mlx5-mod
--with-srp-mod --with-iser-mod --with-isert-mod"
echo
```

Starting the Drivers

1. Start the relevant driver (MLNX_OFED/MLNX_EN):


```
/etc/init.d/openibd restart
```

or

```
systemctl restart openibd.service
```

2. Verify that the supported version of firmware is installed.

```
ibv_devinfo
```

 To configure NVIDIA ConnectX adapter card ports to work with the desired transport, please refer to [Port Type Management/VPI Cards Configuration](#).

Running XLIO

In this page we show how to run a simple network benchmarking test and compare the running results using the kernel's network stack and that of XLIO.

Before running a user application, you must set the library libxlio.so into the environment variable LD_PRELOAD. For further information, please refer to the XLIO User Manual.

Example:

```
$ LD_PRELOAD=libxlio.so sockperf server -i 11.4.3.3
```

⚠ If LD_PRELOAD is assigned with libxlio.so without a path (as in the Example) then libxlio.so is read from a known library path under your distributions' OS otherwise it is read from the specified path.

As a result, an XLIO header message should precede your running application.

```
XLIO INFO : -----  
XLIO INFO : XLIO_VERSION: X.Y.Z-R Release built on MM DD YYYY HH:mm:ss  
XLIO INFO : Cmd Line: sockperf server -i 11.4.3.3  
XLIO INFO : OFED Version: MLNX_OFED_LINUX-X.X-X.X.X.X:  
XLIO INFO : -----  
XLIO INFO : Log Level INFO [XLIO_TRACELEVEL]  
XLIO INFO : -----
```

Followed by the application output, please note that:

- The XLIO version is shown
- The command line indicates your application's name (in the above example: sockperf)

The appearance of the XLIO header indicates that the XLIO library is loaded with your application.

Running Using Non-Root Permission

- Check that ld is able to find the libxlio library:

```
ld -lxlion -verbose
```

- Set UID bit to enforce user ownership:
 - RHEL:

```
sudo chmod u+s /usr/lib64/libxlio*  
sudo chmod u+s /sbin/sysctl
```

- Ubuntu:

```
sudo chmod u+s /usr/lib/libxlio*
sudo chmod u+s /sbin/sysctl
```

- Grant CAP_NET_RAW privileges to application:

```
sudo setcap cap_net_raw,cap_net_admin+ep /usr/bin/sockperf
```

- Launch application under no root:

```
LD_PRELOAD=libxlio.so sockperf sr -i 10.0.0.4 -p 12345
LD_PRELOAD=libxlio.so sockperf pp -i 10.0.0.4 -p 12345 -t10
```

Libxlio-debug.so

libxlio.so is limited to DEBUG log level. In case it is required to run XLIO with detailed logging higher than DEBUG level, use a library called libxlio-debug.so that comes with OFED installation.

Before running your application, set the library libxlio-debug.so into the environment variable LD_PRELOAD (instead of libxlio.so).

Example:

```
$ LD_PRELOAD=libxlio-debug.so sockperf server -i 11.4.3.3
```

⚠ libxlio-debug.so is located in the same library path as libxlio.so under your distribution's OS. For example, in RHEL8.x x86_64, the libxlio.so is located in /usr/lib64/libxlio-debug.so.

⚠ NOTE: If you need to compile XLIO with a log level higher than DEBUG run "configure" with the following parameter:

```
./configure --enable-opt-log=none
```


XLIO Installation Options

Installing XLIO Binary Manually

XLIO can be installed from a dedicated XLIO RPM or a Debian package. In this case, please make sure that the MLXN Driver is already installed and that MLNX Driver and XLIO versions match so that XLIO functions correctly.

This option is suitable for users who receive an OEM XLIO version, or a Fast Update Release XLIO version which is newer than the installed XLIO version.

Installing the XLIO Packages

This section addresses both RPM and DEB (Ubuntu OS) installations.

- The libxlio package contains the binary library shared object file (.so), configuration and documentation files
- The libxlio-utils package contains utilities such as xlio_stats to monitor xlio traffic and statistics
- The libxlio-devel package contains XLIO's extra API header files, for extra functionality not provided by the socket API

Before you begin, please verify the following prerequisites:

- For RPM packages, run:

```
#rpm -qil libxlio
```

- For DEB packages, run:

```
#dpkg -s libxlio
```

If the XLIO packages are installed, the RPM or the DEB logs the XLIO package information and the installed file list. Otherwise, an error message will be displayed.

- Uninstall the current XLIO:
For RPM packages, run:

```
#rpm -e libxlio
```

- For DEB packages:

```
#apt-get remove libxlio
```

Installing the XLIO Binary Package

1. Go to the location where the libxlio package was saved.
2. Run the command below to start installation:
For RPM packages:

```
#rpm -i libxlio-X.Y.Z-R.<arch>.rpm
```

- For DEB packages:

```
#dpkg -i libxlio_X.Y.Z-R_<arch>.deb
```

During the installation process the:

- XLIO library is copied to standard library location (e.g. `/usr/lib64/libxlio.so`). In addition XLIO debug library is copied under the same location (e.g. `/usr/lib64/libxlio-debug.so`)
- README is installed at `/usr/share/doc/libxlio-X.Y.Z/`
- XLIO installs its configuration file, `libxlio.conf`, to the following location: `/etc/libxlio.conf`
- The `xliod` service utility is copied into `/sbin`
- The script `xlio` is installed under `/etc/init.d/` for systems with SysV. This script can be used to load and unload the XLIO service utility.

Install the XLIO utils Package

1. Go to the location where the utils package was saved.
2. Run the command below to start installation:
For RPM packages:

```
#rpm -i libxlio-utils-X.Y.Z-R.<arch>.rpm
```

- For DEB packages:

```
#dpkg -i libxlio-utils_X.Y.Z-R_<arch>.deb
```

During the installation process, the XLIO monitoring utility is installed at `/usr/bin/xlio_stats`.

Installing the XLIO devel Package

1. Go to the location where the devel package was saved.
2. Run the command below to start installation:
For RPM packages:

```
#rpm -i libxlio-devel-X.Y.Z-R.<arch>.rpm
```

- For DEB packages:

```
#dpkg -i libxlio-dev_X.Y.Z-R_<arch>.deb
```

During the installation process the XLIO extra header file is installed at `/usr/include/mellanox/xlio_extra.h`.

Verifying XLIO Installation

For further information, please refer to [Running XLIO](#).

Building XLIO From Sources

XLIO as an open source enables users to try the code, inspect and modify. This option assumes that XLIO (binary package) is already installed on top of NVIDIA® driver, and functions correctly (as explained in [Installing the XLIO Binary Package](#)). Users can download the XLIO sources from [libxlio GitHub](#) and build a new VMA version. This option is suitable for users who wish to implement custom VMA modifications.

For building XLIO from sources, please visit [libxlio GitHub](#)

For adding high debug log level to XLIO, compile it with:

```
./configure --enable-opt-log=none <other configure parameters>
```

Verifying XLIO Compilation

The compiled XLIO library is located under:

```
<path-to-xlio-sources-root-tree>/src/xlio/.libs/libxlio.so
```

When running a user application, you must set the compiled library into the environment variable LD_PRELOAD.

Example:

```
#LD_PRELOAD=<path-to-xlio-sources-root-tree>/src/xlio/.libs/libxlio.so sockperf ping-pong -t 5 -i 224.22.22.22
```

The XLIO banner, mentioned in [Running XLIO](#) section, indicates the library is loaded with the application.

Note: It is recommended to keep the original [libxlio.so](#) under the distribution's original location (e.g. /usr/lib64/[libxlio.so](#)) and not to override it with the newly compiled [libxlio.so](#).

Beta-Installing XLIO as DOCA profile

XLIO can be installed from a dedicated XLIO DOCA profile. In this case, please make sure to follow the [NVIDIA DOCA Installation Guide for Linux - NVIDIA Docs](#).

For XLIO we have a dedicated DOCA profile “doca-xlio”, the profile will install xlio and all its dependencies. To be able to use XLIO there is no need to install any other doca profile except “doca-xlio”.

Please follow the instructions for installing “doca-all” but replace it with “doca-xlio”, for example:

```
"sudo yum install -y doca-xlio"
```

Uninstalling XLIO

Automatic XLIO Uninstallation

If you are about to install a new NVIDIA driver version, the old XLIO version will be automatically uninstalled as part of this process (followed by a new XLIO version installation). Please refer to [Installing XLIO Binary as Part of NVIDIA Drivers](#) for installing NVIDIA drivers command details.

If you are about to uninstall the NVIDIA Driver, XLIO will be automatically uninstalled as part of this process.

Manual XLIO Uninstallation

If you are about to manually uninstall XLIO packages, please run the following:

- For RPM packages:

```
#rpm -e libxlio-utils
#rpm -e libxlio-devel
#rpm -e libxlio
```

- For DEB packages:

```
#dpkg -r libxlio-utils
#dpkg -r libxlio-dev
#dpkg -r libxlio
```

When you uninstall XLIO, the *libxlio.conf* configuration file is saved with the existing configuration. The path of the saved path is displayed immediately after the uninstallation is complete.

Upgrading libxlio.conf

When you upgrade XLIO (through automatic or manual installation), the *libxlio.conf* configuration file is handled as follows:

- If the existing configuration file has been modified since it was installed and is different from the upgraded RPM or DEB, the modified version will be left in place, and the version from the new RPM or DEB will be installed with a new suffix
- If the existing configuration file has not been modified since it was installed, it will automatically be replaced by the version from the upgraded RPM or DEB
- If the existing configuration file has been edited on disk, but is not actually different from the upgraded RPM or DEB, the edited version will be left in place; the version from the new RPM or DEB will not be installed

Type Management / VPI Cards Configuration

NVIDIA ConnectX-6, ConnectX-7 and BlueField-3 ports can be individually configured to work as InfiniBand or Ethernet ports. By default, ConnectX-6/ConnectX-7/BlueField-3 ports are initialized as InfiniBand ports. If you wish to change the port type, use the *mlxconfig* script after the driver is loaded.

For further information on how to set the port type in ConnectX-6/ConnectX-7/BlueField-3, please refer to the MFT User Manual at: <https://docs.nvidia.com/networking/category/mft>.

Basic Performance Tuning

The [Performance Tuning Guide](#) can be used in the XLIO case for detailed instructions on how to optimally tune your machines for XLIO performance.

Binding XLIO to the Closest NUMA

1. Check which NUMA is related to your interface.

```
cat /sys/class/net/<interface_name>/device/numa_node
```

Example:

```
[root@r-host142 ~]# cat /sys/class/net/ens5/device/numa_node  
1
```

The output above shows that your device is installed next to NUMA

2. Check which CPU is related to the specific NUMA.
The output above shows that:
 - CPUs 0-13 & 28-41 are related to NUMA 0
 - CPUs 14-27 & 42-55 are related to NUMA 1Since we want to use NUMA 1, one of the following CPUs should be used: 14-27 & 42-55
3. Use the "taskset" command to run the XLIO process on a specific CPU.

- Server-Side:


```
LD_PRELOAD=libxl原因.so taskset -c 15 sockperf sr -i < IP of FIRST machine MLX interface >
```

- Client-Side:

```
LD_PRELOAD=libxlio.so taskset -c 15 sockperf pp -i < IP of FIRST machine MLX interface >
```

In this example, we use CPU 15 that belongs to NUMA 1. You can also use "numactl - -hardware".

Configuring the BIOS

 Each machine has its own BIOS parameters. It is important to implement any server manufacturer and Linux distribution tuning recommendations for lowest latency.

When configuring the BIOS, please pay attention to the following:

1. Enable Max performance mode.
2. Enable Turbo mode.
3. Power modes - disable C-states and P-states, do not let the CPU sleep on idle.
4. Hyperthreading - there is no right answer if you should have it ON or OFF.
 - ON means more CPU to handle kernel tasks, so the amortized cost will be smaller for each CPU
 - OFF means do not share cache with other CPUs, so cache utilization is betterIf all of your system jitter is under control, it is recommended to turn it OFF, if not keep it ON.
5. Disable SMI interrupts.
Look for "Processor Power and Utilization Monitoring" and "Memory Pre-Failure Notification" SMIs.
The OS is not aware of these interrupts, so the only way you might be able to notice them is by reading the CPU msr register.

Please make sure to carefully read your vendor BIOS tuning guide as the configuration options differ per vendor.

XLIO Configuration

You can control the behavior of XLIO by configuring:

- The *libxlio.conf* file
- XLIO configuration parameters, which are Linux OS environment variables
- XLIO extra API

Configuring libxlio.conf

The installation process creates a default configuration file, */etc/libxlio.conf*, in which you can define and change the following settings:

- The target applications or processes to which the configured control settings apply. By default, XLIO control settings are applied to all applications.
- The transport to be used for the created sockets.
- The IP addresses and ports in which you want offload.

By default, the configuration file allows XLIO to offload everything except for the DNS server-side protocol (UDP, port 53) which will be handled by the OS.

In the *libxlio.conf* file:

- You can define different XLIO control statements for different processes in a single configuration file. Control statements are always applied to the preceding target process statement in the configuration file.
- Comments start with # and cause the entire line after it to be ignored.
- Any beginning whitespace is skipped.
- Any line that is empty is skipped.
- It is recommended to add comments when making configuration changes.

The following sections describe configuration options in *libxlio.conf*. For a sample *libxlio.conf* file, see [Example of XLIO Configuration](#) below.

Configuring Target Application or Process


The target process statement specifies the process to which all control statements that appear between this statement and the next target process statement apply.

Each statement specifies a matching rule that all its sub-expressions must evaluate as true (logical and) to apply.

If not provided (default), the statement matches all programs.

The format of the target process statement is:

```
application-id <program-name|*> <user-defined-id|*>
```

Option	Description
<program-name *>	Define the program name (not including the path) to which the control statements appearing below this statement apply. Wildcards with the same semantics as "ls" are supported (* and ?). For example: <ul style="list-style-type: none">• db2* matches any program with a name starting with db2.• t?cp matches tcp, etc.
<user-defined-id *>	Specify the process ID to which the control statements appearing below this statement apply. <div style="border: 1px solid yellow; padding: 5px; margin-top: 10px;"> You must also set the XLIO_APPLICATION_ID environment variable to the same value as <i>user-defined-id</i>.</div>

Configuring Socket Transport Control

Use socket control statements to specify when libxlio will offload AF_INET/SOCK_STREAM or AF_INET/SOCK_DGRAM sockets (currently SOCK_RAW is not supported).

Each control statement specifies a matching rule that all its sub-expressions must evaluate as true (logical and) to apply. Statements are evaluated in order of definition according to "first-match".

Socket control statements use the following format:

```
use <transport> <role> <address|*>:<port range|*>
```

Where:

Option	Description
transport	Define the mode of transport: <ul style="list-style-type: none"> • xlio - XLIO should be used. • os - the socket should be handled by the OS network stack. In this mode, the sockets are not offloaded. The default is <i>xlio</i> .
role	Specify one of the following roles: <ul style="list-style-type: none"> • tcp_server - for listen sockets. Accepted sockets follow listen sockets. Defined by local_ip:local_port. • tcp_client - for connected sockets. Defined by remote_ip:remote_port:local_ip:local_port • udp_sender - for TX flows. Defined by remote_ip:remote_port • udp_receiver - for RX flows. Defined by local_ip:local_port • udp_connect - for UDP connected sockets. Defined by remote_ip:remote_port:local_ip:local_port
address	You can specify the local address the server is bind to or the remote server address the client connects to. The syntax for address matching is: <IPv4 address>[/<prefix_length>]* <ul style="list-style-type: none"> • IPv4 address - [0-9]+\.[0-9]+\.[0-9]+\.[0-9]+ each sub number < 255 • prefix_length - [0-9]+ and with value <= 32. A prefix_length of 24 # matches the subnet mask 255.255.255.0 . A prefix_length of 32 requires matching of the exact IP.
port range	Define the port range as: <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <code>start-port [-end-port]</code> </div> Port range: 0-65536


Example of XLIO Configuration

To set the following:

- Apply the rules to program *tcp_lat* with ID *B1*
- Use XLIO by TCP clients connecting to machines that belong to subnet *192.168.1.**
- Use OS when TCP server listens to port *5001* of any machine

In *libxlio.conf*, configure:


```
application-id tcp-lat B1
use xlio tcp_client 192.168.1.0/24:*:*:*
use os tcp_server *:5001
use os udp_connect *:53
```

 You must also set the XLIO parameter:

```
XLIO_APPLICATION_ID=B1
```

XLIO Configuration Parameters

XLIO configuration is performed using environment variables. For the full list of XLIO parameters, please see [libxlio README file](#).

 XLIO parameters must be set prior to loading the application with XLIO. You can set the parameters in a system file, which can be run manually or automatically.

All the parameters have defaults that can be modified.

On default startup, the XLIO library prints the XLIO version information, as well as the configuration parameters being used and their values to stderr.

XLIO always logs the values of the following parameters, even when they are equal to the default value:

- XLIO_TRACELEVEL
- XLIO_LOG_FILE


For all other parameters, XLIO logs the parameter values only when they are not equal to the default value.

Beta Level Features Configuration Parameters

The following table lists configuration parameters and their possible values for new XLIO Beta level features. The parameters below are disabled by default.

These XLIO features are still experimental and subject to changes. They can help improve performance of multithread applications.

We recommend altering these parameters in a controlled environment until reaching the best performance tuning.

XLIO Configuration Parameter	Description and Examples
XLIO_RING_MIGRATION_RATIO_TX XLIO_RING_MIGRATION_RATIO_RX	Ring migration ratio is used with the "ring per thread" logic in order to decide when it is beneficial to replace the socket's ring with the ring allocated for the current thread. Each XLIO_RING_MIGRATION_RATIO iteration (of accessing the ring), the current thread ID is checked to see whether the ring matches the current thread. If not, ring migration is considered. If the ring continues to be accessed from the same thread for a certain iteration, the socket is migrated to this thread ring. Use a value of -1 in order to disable migration. Default: -1
XLIO_RING_LIMIT_PER_INTERFACE	Limits the number of rings that can be allocated per interface. For example, in ring allocation per socket logic, if the number of sockets using the same interface is larger than the limit, several sockets will share the same ring. <div data-bbox="922 1066 2040 1166" style="border: 1px solid #f0e68c; padding: 5px; margin: 10px 0;">  XLIO_RX_BUFS might need to be adjusted in order to have enough buffers for all rings in the system. Each ring consumes XLIO_RX_WRE buffers. </div> Use a value of 0 for an unlimited number of rings. Default: 0 (no limit)

XLIO Configuration Parameter	Description and Examples
XLIO_RING_DEV_MEM_TX	<p>XLIO can use the on-device-memory to store the egress packet if it does not fit into the BF inline buffer. This improves application egress latency by reducing the PCI transactions.</p> <p>Using XLIO_RING_DEV_MEM_TX, enables the user to set the amount of the on-device-memory buffer allocated for each TX ring.</p> <p>The total size of the on-device-memory is limited to 256k for a single port HCA and to 128k for dual port HCA. Default value is 0</p>
XLIO_TCP_CC_ALGO	<p>TCP congestion control algorithm.</p> <p>The default algorithm coming with LWIP is a variation of Reno/New-Reno.</p> <p>The new Cubic algorithm was adapted from FreeBSD implementation.</p> <p>Use value of 0 for LWIP algorithm.</p> <p>Use value of 1 for the Cubic algorithm.</p> <p>Use value of 2 in order to disable the congestion algorithm.</p> <p>Default: 0 (LWIP).</p>

Loading XLIO Dynamically

XLIO can be loaded using Dynamically Loaded (DL) libraries. These libraries are not automatically loaded at program link time or start-up as with LD_PRELOAD. Instead, there is an API for opening a library, looking up symbols, handling errors, and closing the library.

The example below demonstrates how to load socket() function. Similarly, users should load all other network-related functions as declared in [sock-redirect.h](#):

```
#include <stdlib.h>
#include <stdio.h>
#include <dlfcn.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef int (*socket_fptr_t) (int __domain, int __type, int __protocol);

int main(int argc, const char** argv)
{
```

```

void* lib_handle;
socket_fptr_t xlio_socket;
int fd;

lib_handle = dlopen("libxlio.so", RTLD_LAZY);
if (!lib_handle) {
    printf("FAILED to load libxlio.so\n");
    exit(1);
}

xlio_socket = (socket_fptr_t)dlsym(lib_handle, "socket");
if (xlio_socket == NULL) {
    printf("FAILED to load socket()\n");
    exit(1);
}

fd = xlio_socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (fd < 0) {
    printf("FAILED open socket()\n");
    exit(1);
}

printf("socket creation succeeded fd = %d\n", fd);
close(fd);
dlclose(lib_handle);
return 0;
}

```

For more information, please refer to *dlopen* man page.

For a complete example that includes all the necessary functions, see [sockperf's xlio-redirect.h](#) and [xlio_socket-redirect.cpp](#) files.

Advanced Features

Direct Packet Control Plane (DPCP)

DPCP provides a unified, flexible interface for programming NVIDIA NICs. DPCP is a prerequisite for enabling XLIO features such as LRO offload, Striding-RQ and TLS HW offload.

XLIO, which comes as part of OFED uses DPCP. Please check the Important Notes under [Changes and New Features](#) for the minimal DPCP version required.

DPCP is an open source project - see its repository [here](#).

TLS HW Offload

TLS HW offload feature accelerates TLS encryption/decryption.

Prerequisites

- Please refer to [System Requirements](#)
- The card must be crypto enabled based on supported cards
- Linux distribution with kTLS support
- Application or TLS library with kTLS support
- OpenSSL library for symmetric encryption SW fallback.

Usage

XLIO offloads Linux kTLS API. Refer to Linux documentation for description of the kTLS API: <https://www.kernel.org/doc/html/latest/networking/tls.html>.

TLS HW offload can be provided to application implicitly by a TLS library with kTLS support such as OpenSSL.

If TLS HW offload cannot be provided setsockopt() syscall returns an error with errno=ENOPROTOOPT. TLS HW offload can be disabled forcibly per direction using configuration options XLIO_UTLS_TX and XLIO_UTLS_RX.

TLS HW offload feature adds new statistics counters. Their presence indicate that offload is configured and works. xlio_stats tool with option -v3 shows TLS statistics for TCP sockets and Rings:

```
=====
      Fd=[59]
- TCP, Non-blocked
- Local Address  = [14.212.1.34:443]
- Foreign Address = [14.212.1.57:49072]
Tx Offload: 18511 / 39409 / 0 / 0 [kilobytes/packets/eagains/errors]
Rx Offload: 1045354 / 2210387 / 0 / 1 [kilobytes/packets/eagains/errors]
Rx byte: cur 0 / max 313 / dropped 0 / limit 0
Rx pkt : cur 0 / max 1 / dropped 0
TLS Offload: version 0303 / cipher 51 / TX On / RX On
TLS Tx Offload: 17394 / 39407 [kilobytes/records]
TLS Rx Offload: 982755 / 2210381 / 28 / 0 [kilobytes/records/encrypted/mixed]
TLS Rx Resyncs: 1 [total]
=====
      RING_ETH=[0]
Tx Offload:      18519 / 39559 [kilobytes/packets]
Rx Offload:      5080 / 39419 [kilobytes/packets]
TLS TX Context Setups: 1
TLS RX Context Setups: 1
Interrupts:      39324 / 38656 [requests/received]
Moderation:      1024 / 1024 [frames/usec period]
=====
```

Description of the statistics counters:

- TLS Offload (version) - 0303 for TLS1.2 and 0304 for TLS1.3.
- TLS Offload (cipher) - 51 for AES128-GCM and 52 for AES256-GCM.
- TLS Offload (TX|RX) - On|Off values turn TLS transmit(TX) and receive(RX) On or Off.
- TLS Tx Offload (kilobytes) - number of offloaded kilobytes excluding headers and other TLS record overhead.
- TLS Tx Offload (records) - number of created and queued TLS records.
- TLS Tx Resyncs - number of HW resynchronizations due to out of sequence send operations.
- TLS Rx Offload (kilobytes) - number of bytes received as TLS payload.

- TLS Rx Offload (records) - total number of TLS records received on the socket.
- TLS Rx Offload (encrypted) - number of encrypted TLS records were decrypted in SW by XLIO.
- TLS Rx Offload (mixed) - number of partially decrypted TLS records handled by XLIO.
- TLS Rx Resyncs - number of times HW loses synchronization.
- TLS TX Context Setups - accumulative counter of created TLS TX contexts what equals to the summary number of sockets with configured TLS TX offload.
- TLS RX Context Setups - accumulative counter of created TLS RX contexts what equals to the summary number of sockets with configured TLS RX offload.

Tuning of XLIO for TLS HW Offload in DNS-over-HTTPS (DoH) scenario

For DNS-over-HTTPS (DoH) scenario there are specific profiles that are optimized for the NGINX frontend side. For x86 server we recommend using XLIO_SPEC=nginx. For NVIDIA DPU system we recommend using XLIO_SPEC=nginx_dpu

Tuning of XLIO for TLS HW Offload in Content Delivery Network (CDN) scenario

The basic profile for Content Delivery Network (CDN) scenario is XLIO_SPEC=nginx.

In the CDN scenario TLS payload often exceeds MTU size. In this case, it is recommended to increase TX buffer size. With larger TX buffers XLIO can create more optimal TLS records.

```
XLIO_TX_BUF_SIZE=16384
```

However, this change may require an increasing number of hugepages configured in the system.

Supported Ciphers

The below table lists all the supported offloaded ciphers.

TLS Version	Bits	Hardware Offload	OpenSSL Name	XLIO Support	
				TX	RX
1.2	128	TLS1.2-AES128-GCM	AES128-GCM-SHA256	YES	YES
			ECDHE-ECDSA-AES128-GCM-SHA256	YES	YES
			ECDHE-RSA-AES128-GCM-SHA256	YES	YES
	256	TLS1.2-AES256-GCM	AES256-GCM-SHA384	YES ¹	YES ¹
			ECDHE-ECDSA-AES256-GCM-SHA384	YES ¹	YES ¹
			ECDHE-RSA-AES256-GCM-SHA384	YES ¹	YES ¹
1.3	128	TLS1.3-AES128-GCM	TLS_AES_128_GCM_SHA256	YES ¹	YES ¹
	256	TLS1.3-AES256-GCM	TLS_AES_256_GCM_SHA384	YES ¹	YES ¹

1. Not supported by RHEL v8.3 yet.

Precision Time Protocol (PTP)

XLIO supports hardware timestamping for UDP-RX flow (only) with Precision Time Protocol (PTP).

When using XLIO on a server running a PTP daemon, XLIO can periodically query the kernel to obtain updated time conversion parameters which it uses in conjunction with the hardware time-stamp it receives from the NIC to provide synchronized time.

Prerequisites

- Support devices: NIC clock
- Set XLIO_HW_TS_CONVERSION environment variable to 4

Usage

1. Set the SO_TIMESTAMPING option for the socket with value SOF_TIMESTAMPING_RX_HARDWARE:

```
uint8_t val = SOF_TIMESTAMPING_RX_HARDWARE
setsockopt(fd, SOL_SOCKET, SO_TIMESTAMPING, &val, sizeof(val));
```

2. Set XLIO environment parameter XLIO_HW_TS_CONVERSION to 4.

Example:

Use the Linux kernel (v4.11) timestamping example found in the kernel source at: *tools/testing/selftests/networking/timestamping/timestamping.c*.

```
Serve
$ sudo LD_PRELOAD=libxlio.so XLIO_HW_TS_CONVERSION=4 ./timestamping <iface> SOF_TIMESTAMPING_RAW_HARDWARE
SOF_TIMESTAMPING_RX_HARDWARE
Client
$ LD_PRELOAD=libxlio.so sockperf tp -i <server-ip> -t 3600 -p 6666 --mps 10
timestamping output:
SOL_SOCKET SO_TIMESTAMPING SW 0.000000000 HW raw 1497823023.070846953 IP_PKTINFO interface index 8
SOL_SOCKET SO_TIMESTAMPING SW 0.000000000 HW raw 1497823023.170847260 IP_PKTINFO interface index 8
SOL_SOCKET SO_TIMESTAMPING SW 0.000000000 HW raw 1497823023.270847093 IP_PKTINFO interface index 8
```

On-Device Memory

Each PCI transaction between the system's RAM and NIC starts at ~300 nsec (and increasing depended on buffer size). Application egress latency can be improved by reducing as many PCI transition as possible on the send path.

Today, XLIO achieves these goals by copying the WQE into the doorbell, and for small packets (<190 Bytes payload) XLIO can inline the packet into the WQE and reduce the data gather PCI transition as well. For data sizes above 190 bytes, an additional PCI gather cycle by the NIC is required to pull the data buffer for egress.

XLIO uses the on-device-memory to store the egress packet if it does not fit into the BF inline buffer. The on-device-memory is a resource managed by XLIO and it is transparent to the user. The total size of the on-device-memory is limited to 256k for a single-port NIC and to 128k for dual-port NIC. Using XLIO_RING_DEV_MEM_TX, the user can set the amount of on-device-memory buffer allocated for each TX ring.

Prerequisites

- Driver: MLNX_OFED version 23.07 and above
- NIC: NVIDIA ConnectX®-6 Dx/ConnectX-7/BlueField-2/BlueField-3 and above
- Protocol: Ethernet.
- Set XLIO_RING_DEV_MEM_TX environment variable to best suit the application's requirements

Verifying On-Device Memory Capability in the Hardware

To verify “On Device Memory” capability in the hardware, run XLIO with DEBUG trace level:

```
XLIO_TRACELEVEL=DEBUG LD_PRELOAD=<path to libxlio.so> <command line>
```

Look in the printout for a positive value of on-device-memory bytes.

For example:

```
Pid: 1748924 Tid: 1748924 XLIO DEBUG : ibch[0x5633333c62f0]:229:print_val() mlx5_2: port(s): 1 vendor: 4125 fw: 22.31.1034 max_qp_wr: 32768 on_device_memory: 131072 packet_pacing_caps: min rate 1, max rate 100000000
Pid: 1748924 Tid: 1748924 XLIO DEBUG : ibch[0x563333340fa60]:229:print_val() mlx5_3: port(s): 1 vendor: 4125 fw: 22.31.1034 max_qp_wr: 32768 on_device_memory: 131072 packet_pacing_caps: min rate 1, max rate 100000000
```


To show and monitor On-Device Memory statistics, run xlio_stats tool.

```
xlio_stats -p <pid> -v 3
```

For example:

```
=====
RING_ETH=[0]
Tx Offload:      858931 / 3402875 [kilobytes/packets]
Rx Offload:      865251 / 3402874 [kilobytes/packets]
Dev Mem Alloc:   16384
Dev Mem Stats:   739074 / 1784935 / 0 [kilobytes/packets/oob]
=====
```

TCP_QUICKACK Threshold

 In order to enable TCP_QUICKACK threshold, the user should modify TCP_QUICKACK_THRESHOLD parameter in the lwip/opt.h file and recompile XLIO.

While TCP_QUICKACK option is enabled, TCP acknowledgments are sent immediately, rather than being delayed in accordance to a normal TCP receive operation. However, sending the TCP acknowledge delays the incoming packet processing to after the acknowledgement has been completed which can affect performance.

TCP_QUICKACK threshold enables the user to disable the quick acknowledgement for payloads that are larger than the threshold. The threshold is effective only when TCP_QUICKACK is enabled, using setsockopt() or using XLIO_TCP_QUICKACK parameter. TCP_QUICKACK threshold is disabled by default.

XLIO Daemon Design

XLIO daemon is responsible for managing all traffic control logic of all XLIO processes, including qdisc, u32 table hashing, adding filters, removing filters, removing filters when the application crashes.

For XLIO daemon usage instructions, refer to the *Installing the XLIO Binary Package* section in the Installation Guide.

TAP Statistics

To show and monitor TAP statistics, run the xlio_stats tool:

```
xlio_stats -p <pid> -v 3
```

Example:

```
=====
      RING_TAP= [0]
Master:      0x29e4260
Tx Offload:  4463 / 67209 [kilobytes/packets]
Rx Offload:  5977 / 90013 [kilobytes/packets]
Rx Buffers:  256
VF Plugouts: 1
Tap fd:      21
Tap Device:  td34f15
=====
      RING_ETH= [1]
Master:      0x29e4260
Tx Offload:  7527 / 113349 [kilobytes/packets]
Rx Offload:  7527 / 113349 [kilobytes/packets]
Retransmissions: 1
=====
```

Output analysis:

- RING_TAP[0] and RING_ETH[1] have the same master 0x29e4260 ring
- 4463 Kbytes/67209 packets were sent from the TAP device
- 5977 Kbytes/90013 packets were received from the TAP device
- Plugout event occurred once
- TAP device fd number was 21, TAP name was *td34f15*

Getting Ring Protection Domain

Pass special structure as an argument into `getsockopt()` with `SO_XLIO_PD` to get protection domain information from ring used for current socket. This information can be available after setting connection for TX ring and bounding to device for RX ring. By default getting PD for TX ring. This case can be used with `sendmsg(SCM_XLIO_PD)` when the data portion contains an array of the elements with datatype as `struct xlio_pd_key`. Number of elements in this array should be equal to `msg_iovlen` value. Every data pointer in `msg_iov` has correspondent memory key.

```
struct xlio_pd_attr {
    uint32_t flags;
    void*     ib_pd;
};
```

NVME over TCP DIGEST Offload Tx (Alpha Level)

NVME over TCP (NVMeoTCP) hardware offload feature accelerates NVMeoTCP DIGEST calculation for transmitted NVME PDUs.

Prerequisites

- Please refer to [System Requirements](#)
- The card must support NVMeoTCP offload (ConnectX-7, Bluefield-3)

Usage

1. Use the application to call `setsockopt (fd, IPPROTO_TCP, TCP_ULP, "nvme", 4)`
2. Call: `setsockopt(fd, NVDA_NVME, NVME_TX, &configure, sizeof(configure))`
where: `uint32_t configure = XLIO_NVME_HDGST_ENABLE | XLIO_NVME_DDGST_ENABLE | XLIO_NVME_DDGST_OFFLOAD`
Note: If any of the `setsockopt` calls fail, offload is not supported.

3. Call: `setsockopt(fd, SOL_SOCKET, SO_ZEROCOPY, &opt_val, sizeof(opt_val))`
where: `int opt_val = 1`
4. Use the application to register TX buffers with PD from XLIO - see section Getting Ring Protection Domain section above
5. Call `sendmsg(fd, msg, MSG_ZEROCOPY)` with extended zero-copy API
where `msg` is of type `msg_hdr`
 - a. With `msg_hdr *cmsg = CMSG_FIRSTHDR(msg);`
 - b. `cmsg->cmsg_level = SOL_SOCKET;`
 - c. `cmsg->cmsg_type = SCM_XLIO_NVME_PD;`
 - d. `cmsg->cmsg_len = msg->msg_controllen;`
 - e. With `CMSG_DATA(cmsg)` set to `xlio_pd_key`

See Kernel TX Zero Copy documentation: https://www.kernel.org/doc/html/v4.15/networking/msg_zerocopy.html

Full examples can be found in XLIO GIT repository: <https://github.com/Mellanox/libxlio>

SO_XLIO_ISOLATE option

Offloaded TCP sockets support `SO_XLIO_ISOLATE` option on `SOL_SOCKET` level. The option allows grouping sockets with specific policy. Value for the option has type 'int' and contains the policy.

Supported policies:

- `SO_XLIO_ISOLATE_DEFAULT` - default behavior according to XLIO configuration.
- `SO_XLIO_ISOLATE_SAFE` - isolate sockets from the default sockets and guarantee thread safety regardless of XLIO configuration. This policy is effective in `XLIO_TCP_CTL_THREAD=delegate` configuration. Socket API thread safety model is not changed.

Limitations:

- `SO_XLIO_ISOLATE` option may be called after `socket()` syscall and before either `listen()` or `connect()`.

Using sockperf with XLIO

Socketperf is XLIO's sample application for testing latency and throughput over a socket API. The precompiled sockperf binary is located in `/usr/bin/sockperf`.

For detailed instructions on how to optimally tune your machines for XLIO performance, please see the [Tuning Guide](#) and [XLIO Performance Tuning Guide](#).

- To run a sockperf UDP throughput test:
 - To run the server, use:

```
LD_PRELOAD=libxlio.so sockperf sr -i <server ip> --msg-size=1472
```

- To run the client, use:

```
LD_PRELOAD=libxlio.so sockperf tp -i <server ip> --msg-size=1472
```

- To run a sockperf TCP throughput test:
 - To run the server, use:

```
XLIO_STRQ=regular_rq LD_PRELOAD=libxlio.so sockperf sr -i <server ip> --msg-size=1472 --tcp
```

- To run the client, use:

```
XLIO_STRQ=regular_rq LD_PRELOAD=libxlio.so sockperf tp -i <server ip> --msg-size=1472 --tcp
```

For more information, please refer to sockperf help using `sockperf -h`.

Example - Running sockperf Ping-pong Test

For optimal performance, please refer to [Basic Performance Tuning](#).

1. Run sockperf server on Host A:

```
LD_PRELOAD=libxlio.so sockperf sr
```

1. Run sockperf client on Host B:

```
LD_PRELOAD=libxlio.so sockperf pp -i <server_ip>
```

XLIO Extra API

Overview of the XLIO Extra API

The information in this chapter is intended for application developers that want to maximize XLIO performance may use the Extra API and achieve the following:

- To further lower latencies
- To increase throughput
- To gain additional CPU cycles for the application logic
- To better control XLIO offload capabilities

All socket applications are limited to the given Socket API interface functions.

The XLIO Extra API enables XLIO to open a new set of functions which allow the application developer to add code which utilizes zero copy receive function calls and low-level packet filtering by inspecting the incoming packet headers or packet payload at a very early stage in the processing.

XLIO is designed as a dynamically linked user-space library. As such, the XLIO Extra API has been designed to allow the user to dynamically load XLIO and to detect at runtime if the additional functionality described here is available or not. The application is still able to run over the general socket library without XLIO loaded as it did previously, or can use an application flag to decide which API to use: Socket API or XLIO Extra API.

The XLIO Extra APIs are provided as a header with the XLIO binary rpm. The application developer needs to include this header file in his application code.

After installing the XLIO RPM on the target host, the XLIO Extra APIs header file is located in the following link:

```
#include "/usr/include/mellanox/xlio_extra.h"
```

The xlio_extra.h provides detailed information about the various functions and structures, and instructions on how to use them.

An example using the XLIO Extra API can be seen in the udp_lat source code. Follow the '--xliozcopyread' flag for the zero copy recvfrom logic.

A specific example for using the TCP zero copy extra API can be seen under extra_api_tests/tcp_zcopy_cb.

Using XLIO Extra API

During runtime, use the `xlio_get_api()` function to check if XLIO is loaded in your application and if XLIO Extra API is accessible.

If the function returns with NULL, either XLIO is not loaded with the application, or the XLIO Extra API is not compatible with the header function used for compiling your application. NULL will be the typical return value when running the application on native OS without XLIO loaded. On success the function returns a valid `api()` pointer and NULL on failure.

Any non-NULL return value is a `xlio_api_t` type structure pointer that holds pointers to the specific XLIO Extra API function calls which are needed for the application to use. Available functions can be checked using special bit mask field as `cap_mask`.

It is recommended to call `xlio_get_api()` once on startup, and to use the returned pointer throughout the life of the process.

There is no need to 'release' this pointer in any way.

Control Off-load Capabilities During Run-Time

Adding libxlio.conf Rules During Run-Time

Adds a `libxlio.conf` rule to the top of the list. This rule will not apply to existing sockets which already considered the conf rules. (around `connect/listen/send/rcv ..`)

Syntax:

```
int (*add_conf_rule)(char *config_line);
```

Return value:

- 0 on success
- error code on failure

Where:

- `config_line`
 - Description - new rule to add to the top of the list (highest priority)
 - Value - a char buffer with the exact format as defined in `libxlio.conf`, and should end with `'\0'`

Creating Sockets as Offloaded or Not-Offloaded

Creates sockets on pthread tid as off-loaded/not-off-loaded. This does not affect existing sockets. Offloaded sockets are still subject to `libxlio.conf` rules.

Usually combined with the `XLIO_OFFLOADED_SOCKETS` parameter.

Syntax:

```
int (*thread_offload)(int offload, pthread_t tid);
```

Return value:

- 0 on success
- error code on failure

Where:

- `offload`
 - Description - Offload property
 - Value - 1 for offloaded, 0 for not-offloaded
- `tid`
 - Description - thread ID

Zero Copy `recvfrom()`

Zero-copy `recvfrom` implementation. This function attempts to receive a packet without doing data copy.

Syntax:

```
int (*recvfrom_zcopy)(int s, void *buf, size_t len, int *flags, struct sockaddr *from, socklen_t *fromlen);
```

Where:

Parameter Name	Description	Values
s	Socket file descriptor	
buf	Buffer to fill with received data or pointers to data (see below).	
flags	Pointer to flags (see below).	Usual flags to <code>recvmsg()</code> , and <code>MSG_XLIO_ZCOPY_FORCE</code>
from	If not NULL, is set to the source address (same as <code>recvfrom</code>)	
fromlen	If not NULL, is set to the source address size (same as <code>recvfrom</code>).	

The flags parameter can contain the usual flags to `recvmsg()`, and also the `MSG_XLIO_ZCOPY_FORCE` flag. If the latter is not set, the function reverts to data copy (i.e., zero-copy cannot be performed). If zero-copy is performed, the flag `MSG_XLIO_ZCOPY` is set upon exit.

If zero copy is performed (`MSG_XLIO_ZCOPY` flag is returned), the buffer is filled with a `xlio_recvfrom_zcopy_packets_t` structure holding as much fragments as `len` allows. The total size of all fragments is returned. Otherwise, the buffer is filled with actual data, and its size is returned (same as `recvfrom()`).

If the return value is positive, data copy has been performed. If the return value is zero, no data has been received.

Freeing Zero Copied Packet Buffers

Frees a packet received by "`recvfrom_zcopy()`" or held by "receive callback".

Syntax:

```
int (*recvfrom_zcopy_free_packets)(int s, struct xlio_recvfrom_zcopy_packet_t *pkts , size_t count);
```

Where:

- s - socket from which the packet was received
- pkts - array of packet identifiers
- count - number of packets in the array

Return value:

- 0 on success, -1 on failure
- errno is set to:
 - EINVAL - not a offloaded socket
 - ENOENT - the packet was not received from 's'.

Example:

```
entry Source Source-mask Dest Dest-mask Interface Service Routing Status Log
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
1 any any any any if0 any tunneling active 1 2 192.168.2.0 255.255..255.0 any any if1 any tunneling active 1
```

Expected result:

```
sRB-20210G-61f0(statistic)# log show counter tx total pack tx total byte rx total pack rx total byte
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
1 2733553 268066596 3698 362404
```


Parameter	Description
tx total byte	The number of transmit bytes associated with a TFM rule; has a log counter n. The above example shows the number of bytes sent from Infiniband to Ethernet (one way) or sent between InfiniBand and Ethernet and matching the two TFM rules with log counter #1.
rx total pack	The number of receive packets associated with a TFM rule; has a log counter n.
rx total byte	The number of receive bytes associated with a TFM rule; has a log counter n.

SocketXtreme API

The XLIO SocketXtreme API has been developed to optimize the data path of the socket API, while preserving the familiar standard socket API for control operations, such as `select()`, `poll()`, `epoll_wait()`, `recv()`, `recvfrom()`, `recvmsg()`, `read()`, and `readv()`.

The XLIO SocketXtreme API enhances application performance in the following ways:

1. **Reduced Context Switching:** The lightweight library call to `socketxtreme_poll(...)` eliminates the need for traditional methods like `poll()`, `select()`, `epoll()`, and similar interfaces, as well as subsequent `read()`, `recv()`, `recvmsg()`, and other socket-related system calls. It achieves asynchronous I/O for both data reception (RX) and data transmission (TX) by concurrently polling multiple sockets on the same ring (more information about rings is provided in subsequent sections).
2. **Comprehensive Data Handling:** The `socketxtreme_poll` function provides the user application with detailed data in the form of the struct `xlio_socketxtreme_completion_t`. This structure, elaborated upon below, equips the application to effectively manage the status of sockets associated with the ring file descriptor and process the data received through these sockets.

Usage

Verify SocketXtreme support

To employ the XLIO extra API and the SocketXtreme interface, follow these steps:

1. Obtain the XLIO API using the `xlio_get_api` function.
2. Verify the `XLIO_MAGICNUMBER` to ensure compatibility.
3. Verify the XLIO capabilities, as demonstrated below.

```
#include <mellanox/xlio extra .h>

static struct xlio_api_t *get_xlio_api (void)
{
    struct xlio_api_t *api_ptr = NULL; int err = xlio_get_api ();
    if (err < 0) {
        return NULL;
    }
    if (api_ptr == NULL) {
        return NULL;
    }
    if (api_ptr->magic != XLIO_MAGICNUMBER) {
        printf ("Unexpected XLIO AP! magic number : expected %"
                PRi64 ", got %" PRi64 "\n",
                (uint64_t)XLIO_MAGICNUMBER , api_ptr->magic);
        goto failed ;
    }

    uint64_t required_caps = XLIO_EXTRA_API_GET_SOCKET_RINGS_FDS |
        XLIO_EXTRA_API_SOCKETXTREME_POLL |
        XLIO_EXTRA_API_SOCKETXTREME_FREE_PACKETS |
        XLIO_EXTRA_API_IOCTL;
    if ((api_ptr->cap_mask & required_caps) != required_caps) {
        printf ("Required XLIO caps are missing: required %" PRi64 ", got %" PRi64 "\n",
                required_caps, api_ptr->cap_mask );
    }
}
```

```

        goto failed ;
    }
    return api_ptr ;
failed :
    free (api_ptr);
    return NULL;
}

```

Replace the socket file descriptor with alternative identifier

XLIO introduces the capability to substitute a socket file descriptor with user-defined data. To utilize this feature, XLIO provides a new setsockopt option, as illustrated in the example below.

```

uint64_t user_data = (uint64_t)user_ptr;
int re = xlio_api->setsockopt (socket_fd, SOL_SOCKET, SO_XLIO_USER_DATA ,
                             &user_data , sizeof (user_data));

if ( re != 0 ) {
    printf ("Failed to set socket user data for sock %d: re %d, errno %d\n", socket_fd, re, errno);
    goto fail;
}

```

Get ring file descriptors(s) from socket file descriptor

The ring file descriptor(s) will be required as arguments to socktxxtreme_poll.

```

int ring_fds[2];
int num_rings;

num_rings = xlio_api->get_socket_rings_fds (socket_fd, ring_fds, 2);
if (num_rings < 0)
    {printf ("Failed to get ring FDs for socket errno %d: rc %d, errno %d\n",
           socket_fd, num_rings, errno);
    }

```

Parameter/Return value	Description
num_rings	The actual number of rings returned by the function. If the number is -1 the function failed check the errno
ring_fds	An array of integers to be filled upon success
num_ring_fds	The maximal number of ring descriptors to fill

Poll ring fd

The ring file descriptor(s) will be required as arguments to `socketxtreme_poll`.

```
struct xlio_socketxtreme_completion_t comps[11AX_EVENTS_PER_POLLJ ;
int num_events = xlio_api->socketxtreme_poll(ring_fd, comps,
MAX_EVENTS_PER_POLL, SOCKETXTREME_POLL_TX) ;
```

Parameter	Description
num_events	The actual number of events returned by the function. If the number is -1 the function failed check the errno
ring_fd	The ring file descriptor to poll
comps	An array of XLIO completion events

Processing the completions

For detailed `xlio_socket_xtreme_completion_t` please review the `xlio_extra.h` header file.

When the `XLIO_SOCKETXTREME_PACKET` flag is enabled within the `xlio_completion_t.events` field, it indicates that the completion is associated with the descriptor of an incoming packet. You can access this descriptor through the `xlio_completion_t.packet` field. This descriptor points to XLIO buffers that hold data distributed by the hardware, ensuring that the data is delivered to the application without the need for copying. It's essential to remember that after the application has finished using the returned packets and their associated buffers, they must be released using the `free_xlio_packets()` and `free_xlio_buff()` functions. If the `XLIO_SOCKETXTREME_PACKET` flag is disabled, the `xlio_completion_t.packet` field remains reserved.

In addition to indicating the arrival of a packet, XLIO also reports the `XLIO_SOCKETXTREME_NEW_CONNECTION_ACCEPTED` event and standard `epoll` events through the `xlio_completion_t.events` field. The `XLIO_SOCKETXTREME_NEW_CONNECTION_ACCEPTED` event is triggered when a new connection is accepted by the server. When using `socketxtreme_poll()`, new connections are accepted automatically, and there's no need to explicitly call `accept()` on the listen socket. The `XLIO_SOCKETXTREME_NEW_CONNECTION_ACCEPTED` event pertains to the newly connected or child socket, with `xlio_completion_t.user_data` referring to the child socket. For events other than packet arrival and new connection acceptance, the `xlio_completion_t.events` bitmask is constructed using standard `epoll` API event types. It's important to note that the same completion can report multiple events; for instance, the `XLIO_SOCKETXTREME_PACKET` flag can be enabled alongside `EPOLLOUT` events and more.

```
for (i = 0; i < num_events; i++ {
    struct xlio_socketxtreme_completion_t *comp = &comps[i];
    assert (comp->user_data != 0):

    /* Convert the user data to an application identifier and consume the data.
     * Below we assume that the application converts the identifier to app_sink pointer.
     */
    app_sink_t *app_sink = (app_sink_t *)comp->user_data ;

    if (comp->events & EPOLLERR) {
        app_sink->process_error_cb ();
    }

    if (comp->events & XLIO_SOCKETXTREME_PACKET) {
        app_sink->process_packet (comp->packet):
    }
    if (comp->events & XLIO_SOCKETXTREME_NEW_CONNECTION_ACCEPTED) {
        app_sink->process_accepted_connection (comp->packet):
    }
}
```

```
}
```

Freeing packets and buffers

In the following example, we iterate through the buffers of a specific packet, free them, and then release the packet.

```
static inline void free_xlio_packet (struct xlio_socketxtreme_packet_desc_t
*packet)
{
    assert (packet != NULL);
    while (packet->num_bufs-->
    {
        assert (packet->buff_lst != NULL);
        struct xlio buff t *buff to free = packet->buff_lst;
        packet->buff_lst = packet->buff_lst->next;
        xlio_api->socketxtreme_ free_buff (buff_to_ free);
    }
    xlio_api->socketxtreme_ free_packets(packet, 1);
}
```

Packet Filter Callback API

The packet filter logic allows developers to inspect and dynamically decide whether to keep or drop incoming packets. The user's packet filtering callback follows this prototype:


```
typedef xlio_recv_callback_retval_t (*xlio_recv_callback_t) (int fd, size_t
sz_iov, struct iovec iov[], struct xlio_info_t *xlio_info, void *context);
```

To register this callback function with XLIO, use the `register_recv_callback()` function provided by the XLIO Extra API. If you wish to unregister the callback, simply set the function pointer to `NULL`.

XLIO invokes this callback to inform the application about new incoming packets. This notification occurs after the internal processing of IP and UDP/TCP headers and precedes the queuing of these packets in the socket's receive queue.

The context of the callback is always that of one of the user's application threads that have previously called one of the following socket APIs: `select()`, `poll()`, `epoll_wait()`, `recv()`, `recvfrom()`, `recvmsg()`, `read()`, or `readv()`.

fd	File descriptor of the socket to which this packet refers.
iov	iovector structure array pointer holding the packet received, data buffer pointers, and the size of each buffer.
iov sz	Size of the iov array.
xlio info	Additional information on the packet and socket.
context	User-defined value provided during callback registration for each socket.

 The application is allowed to invoke all Socket APIs from within the callback context. However, it's important to note that depending on how the application behaves in this context, packet loss may occur. In cases where the callback behavior is extremely rapid and non-blocking, it's less likely to lead to packet loss. Regarding the parameters "iov" and "xlio_info," it's crucial to understand that their validity is limited to the duration of the callback context. If you intend to work with zero-copy logic and require these structures for later use, it's advisable to make copies of them before exiting the callback context.

Dump fd Statistics using XLIO Logger

Dumps statistics for fd number using log_level log level.

Syntax:

```
int (*dump_fd_stats) (int fd, int log_level);
```

Parameters:

Parameter	Description
fd	fd to dump, 0 for all open fds.
log_level	log_level dumping level corresponding vlog_levels_t enum (vlogger.h): VLOG_NONE = -1 VLOG_PANIC = 0 VLOG_ERROR = 1 VLOG_WARNING = 2 VLOG_INFO = 3 VLOG_DETAILS = 4 VLOG_DEBUG = 5 VLOG_FUNC = VLOG_FINE = 6 VLOG_FUNC_ALL = VLOG_FINER = 7 VLOG_ALL = 8

For output example see section [Monitoring - the xlio_stats Utility](#). Return values: 0 on success, -1 on failure

"Dummy Send" to Improve Low Message Rate Latency

The “Dummy Send” feature gives the application developer the capability to send dummy packets in order to warm up the CPU caches on XLIO send path, hence minimizing any cache misses and improving latency. The dummy packets reaches the hardware NIC and then is dropped.

The application developer is responsible for sending the dummy packets by setting the XLIO_SND_FLAGS_DUMMY bit in the flags parameter of send(), sendto(), sendmsg(), and sendmmsg() sockets API.

Parameters:


Parameter	Description
XLIO_SND_FLAGS_DUMMY	Indicates a dummy packet

Same as the original APIs for offloaded sockets. Otherwise, -1 is returned and errno is set to EINVAL. Return values:

Usage example:

```
void dummyWait(Timer waitDuration, Timer dummySendCycleDuration) { Timer now = Timer::now(); Timer endTime = now +
waitDuration; Timer nextDummySendTime = now + dummySendCycleDuration; for ( ; now < endTime ; now = Timer::now())
{ if (now >= nextDummySendTime) { send(fd, buf, len, XLIO_SND_FLAGS_DUMMY); nextDummySendTime +=
dummySendCycleDuration; } } }
```

This sample code consistently sends dummy packets every *DummySendCycleDuration* using the XLIO extra API while the total time does not exceed *waitDuration*.

 It is recommended not to send more than 50k dummy packets per second.

Verifying “Dummy Send” Capability in HW

In order to verify “Dummy Send” capability in the hardware, run XLIO with DEBUG trace level.

```
XLIO_TRACELEVEL=DEBUG LD_PRELOAD=<path to libxlio.so> <command line>
```

Look in the printout for “HW Dummy send support for QP = [0|1]”.

For example:

```
Pid: 3832 Tid: 3832 XLIO DEBUG: qpm[0x2097310]:121:configure() Creating QP of transport type 'ETH' on ibv device
'mlx5_0' [0x201e460] on port 1 Pid: 3832 Tid: 3832 XLIO DEBUG: qpm[0x2097310]:137:configure() HW Dummy send
support for QP = 1 Pid: 3832 Tid: 3832 XLIO DEBUG: cqm[0x203a460]:269:cq_mgr() Created CQ as Tx with fd[25] and of
size 3000 elements (ibv_cq_hndl=0x20a0000)
```

“Dummy Packets” Statistics

Run *xlio_stats* tool to view the total amount of dummy-packets sent.

```
xlio_stats -p <pid> -v 3
```

The number of dummy messages sent will appear under the relevant fd. For example:

```
===== Fd=[20] - UDP, Blocked, MC Loop Enabled - Local Address = [0
.0.0.0:56732] Tx Offload: 128 / 9413 / 0 / 0 [kilobytes/packets/drops/errors] Tx Dummy messages : 87798 Rx
Offload: 128 / 9413 / 0 / 0 [kilobytes/packets/eagains/errors] Rx byte: cur 0 / max 14 / dropped 0 / limit 212992
Rx pkt : cur 0 / max 1 / dropped 0 Rx poll: 0 / 9411 (100.00%) [miss/hit]
=====
```

Control the Library

This function allows to communicate with library using extendable protocol

based on struct cms_hdr.

Syntax:

```
int (*ioctl) (void *cms_hdr, size_t cms_len);
```

Parameters:

Parameter	Description
cms_hdr	The address of the ancillary data.
cms_len	The length of the ancillary data is passed in cms_hdr. Note that if multiple ancillary data sections are being passed, this length should reflect the total length of ancillary data sections

The `msg_hdr` parameter points to the ancillary data. This `msg_hdr` pointer points to the following structure (C/C++ example shown) that describes the ancillary data.

```
struct cmsghdr {
    size_t    cmsg_len;      /* data byte count includes hdr */
    int      cmsg_level;    /* originating protocol */
    int      cmsg_type;     /* protocol-specific type */
    /* followed by u_char    cmsg_data[]; */
};
```

Ancillary data is a sequence of `cmsghdr` structures with appended data. The sequence of `cmsghdr` structures should never be accessed directly. Instead, use only the following macros: `MSG_ALIGN`, `MSG_SPACE`, `MSG_DATA`, `MSG_LEN`.

Guidelines:

- The `msg_len` should be set to the length of the `cmsghdr` plus the length of all ancillary data that follows immediately after the `cmsghdr`. This is represented by the commented out `msg_data` field.
- The `msg_level` should be set to the option level (for example, `SOL_SOCKET`).
- The `msg_type` should be set to the option name (for example, `MSG_XLIO_IOCTL_USER_ALLOC`).

Supported commands:

Command	Description
<code>MSG_XLIO_IOCTL_USER_ALLOC</code>	Use user defined function to allocate global pools

CMSG_XLIO_IOCTL_USER_ALLOC

Filed size	Description
uint8_t	control flags
uintptr_t	pointer to memory allocation function
uintptr_t	pointer to memory free function

Control Flags

```
enum {  
    IOCTL_USER_ALLOC_TX = (1 << 0),  
    IOCTL_USER_ALLOC_RX = (1 << 1),  
    IOCTL_USER_ALLOC_TX_ZC = (1 << 2)  
};
```

Usage Example

In this example, the application uses CMSG_XLIO_IOCTL_USER_ALLOC command.

```
#include <sys/socket.h>  
#include <sys/types.h>  
#include <netinet/in.h>  
#include <netdb.h>  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <unistd.h>
```

```

#include <errno.h>
#include <arpa/inet.h>
#include <mellanox/xlio_extra.h>
void * my_alloc(size_t sz_bytes)
{
    void *m_data_block = NULL;
    long page_size = sysconf(_SC_PAGESIZE);
    if (page_size > 0) {
        sz_bytes = (sz_bytes + page_size - 1) & (~page_size - 1);
        int ret = posix_memalign(&m_data_block, page_size, sz_bytes);
        if (!ret) {
            return NULL;
        }
    }
    return m_data_block;
}
void my_free(void *ptr)
{
    free(ptr);
}
int main(int argc, char *argv[])
{
    int sockfd = 0, n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;
    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }
#pragma pack(push, 1)
    struct {
        uint8_t flags;
        void* (*alloc_func)(size_t);
        void (*free_func)(void *);
    } data;
#pragma pack(pop)
    struct cmsghdr *cmsg;
    char cbuf[CMSG_SPACE(sizeof(data))];
    errno = 0;
    cmsg = (struct cmsghdr *)cbuf;
    cmsg->cmsg_level = SOL_SOCKET;

```

```

cmsg->cmsg_type = CMSG_XLIO_IOCTL_USER_ALLOC;
cmsg->cmsg_len = CMSG_LEN(sizeof(data));
data.flags = 0x03;
data.alloc_func = my_alloc;
data.free_func = my_free;
memcpy(CMSG_DATA(cmsg), &data, sizeof(data));

struct xlio_api_t *extra_api;
extra_api = xlio_get_api();
printf("extra_api=%p\n", extra_api);

int rc = 0;
if (extra_api) rc = extra_api->ioctl(cmsg, cmsg->cmsg_len);
printf("extra_api->ioctl() rc=%d\n");
memset(recvBuff, '0', sizeof(recvBuff));
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("\n Error : Could not create socket \n");
    return 1;
}

memset(&serv_addr, '0', sizeof(serv_addr));

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(5000);

if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
{
    printf("\n inet_pton error ocured\n");
    return 1;
}
if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    printf("\n Error : Connect Failed \n");
    return 1;
}

while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)
{
    recvBuff[n] = 0;
    if(fputs(recvBuff, stdout) == EOF)
    {

```

```
        printf("\n Error : Fputs error\n");
    }
}
if(n < 0)
{
    printf("\n Read error \n");
}
return 0;
}
```

XLIO Socket API

General Information

XLIO Socket API is an event-based API for the high-performance scenarios. This is a non-standard API and requires the application to be integrated explicitly.

XLIO Socket API triggers a callback immediately when a respective event happens. This reduces latency and simplifies handling of the events. The API also allows to avoid events aggregation if they turn out to be unnecessary.

There are two ways to call the API:

1. Direct function calls: The prototypes are declared in `<mellanox/xlio.h>`. This approach requires explicit linkage with XLIO static library.
2. Indirect function calls by the pointers which are provided by `xlio_get_api()`. The prototypes are declared in `<mellanox/xlio_extra.h>`.

Common types are defined in `<mellanox/xlio_types.h>`, which is included implicitly by the above headers.

Current limitations:

- Only TCP sockets are supported.
- Only polling mode is supported.
- No listen sockets support.

For a sample application, please refer to tests/extra_api/xlio_socket_api.c within the XLIO sources.

Global Initialization

XLIO Socket API requires explicit global initialization before using any other functions. The initialization is a heavy process and is expected to be performed in advance.

Types definitions

```
struct xlio_init_attr {
    unsigned flags;
    xlio_memory_cb_t memory_cb;
    /* Optional external user allocator for XLIO buffers. */
    void *(*memory_alloc)(size_t);
    void (*memory_free)(void *);
};
```

Where

Field	Description
flags	Global flags. Currently unused
memory_cb	An optional callback called when XLIO allocates memory for data buffers. Zerocopy RX buffers points to such memory only. User can use this information to prepare the allocated memory for further processing of the zerocopy RX data
memory_alloc, memory_free	An optional external allocator to be used for the data buffers. The external allocator and memory_cb are orthogonal and may be used together

Syntax

Global initialization


```
int xlio_init_ex(const struct xlio_init_attr *attr);
```

Where

Argument	Description
attr	Global attributes

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

 User should finalize XLIO library with xlio_exit() when it is no longer needed. Usually, this is done during the termination phase. Both XLIO Socket API and intercepted POSIX API may not be used after the finalization.


Syntax

Global finalization

```
xlio_exit();
```

XLIO Polling Groups

An XLIO polling group is a collection of XLIO sockets and their internal auxiliary objects. An XLIO polling group is represented by the opaque `xlio_poll_group_t` type.

 Polling groups do not share objects. Thus, object migration between groups is not supported.

Operations with different polling groups do not overlap, except for unlikely protected access to global pools. Therefore, multiple polling groups can work in parallel without serialization.

Operations with the same polling group must be serialized.

Polling groups are not bound to CPU/thread. It is allowed to use a single polling group on multiple CPUs if serialization is guaranteed. For example, this approach can be used for a polling group migration implementation.

Recommendations:

- Polling groups are expected to be long-lived objects.
- It is expected to use polling group per CPU/thread and probably a small number of extra groups.
- Each polling group creates HW objects per utilized network interface. Minimizing the number of utilized network interfaces per group will improve HW resources utilization.

A major part of the XLIO activities is done in the context of `xlio_poll_group_poll()` call. Therefore, this function should be called frequently enough to reduce latency and avoid runtime issues such as timeouts and TCP retransmissions.

Flags definitions

```
#define XLIO_GROUP_FLAG_SAFE 0x1
#define XLIO_GROUP_FLAG_DIRTY 0x2
```

Where

Flag	Description
XLIO_GROUP_FLAG_SAFE	Relaxes thread-safety requirements: allows to call a send operation concurrently with the polling group operations. However, all the group operations and socket creation/destruction still must be serialized. Concurrent send operations still must be serialized. This flag has a runtime cost and is expected to be used for performance non-critical sockets
XLIO_GROUP_FLAG_DIRTY	Requests the group to track dirty sockets. Required for xlio_poll_group_flush() to function

Types definitions

```

struct xlio_poll_group_attr {
    unsigned flags;
    void (*socket_event_cb)(xlio_socket_t, uintptr_t userdata_sq, int event, int value);
    void (*socket_comp_cb)(xlio_socket_t, uintptr_t userdata_sq, uintptr_t userdata_op);
    void (*socket_rx_cb)(xlio_socket_t, uintptr_t userdata_sq, void *data, size_t len,
                        struct xlio_buf *buf);
};

```

Where

Field	Description
flags	Polling group flags
socket_event_cb	Mandatory callback for socket events
socket_comp_cb	Completion callback for zerocopy send operations
socket_rx_cb	Callback for RX data delivery

Syntax

Creating XLIO polling group

```
int xlio_poll_group_create(const struct xlio_poll_group_attr *attr, xlio_poll_group_t *group_out);
```

Where

Argument	Description
attr	Polling group attributes
group_out	On success, the created polling group is saved there

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

Syntax

XLIO polling group destruction

```
int xlio_poll_group_destroy(xlio_poll_group_t group);
```

Where

Argument	Description
group	XLIO polling group

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

Syntax

```

Polling

void xlio_poll_group_poll(xlio_poll_group_t group);

```

Where

Argument	Description
group	XLIO polling group

XLIO Sockets

XLIO socket is similar to the POSIX socket, except it has a separate non-overlapping API. An XLIO socket is represented by the opaque `xlio_socket_t` type.

XLIO sockets have the following properties:

- Always non-blocking.
- No partial write support. Either all the data is accepted, or the call fails.

Types definitions

```
struct xlio_socket_attr {
    unsigned flags;
    int domain; /* AF_INET or AF_INET6 */
    xlio_poll_group_t group;
    uintptr_t userdata_sq;
};
```

Where

Field	Description
flags	Socket flags, currently unused
domain	Address family: either AF_INET or AF_INET6
group	XLIO polling group
userdata_sq	Opaque per-socket userdata

Syntax

XLIO socket creation

```
int xlio_socket_create(const struct xlio_socket_attr *attr, xlio_socket_t *sock_out);
```

Where

Argument	Description
attr	Socket attributes
sock_out	On success, the created socket object is saved there

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

Syntax

XLIO socket destruction

```
int xlio_socket_destroy(xlio_socket_t sock);
```

Where

Argument	Description
sock	XLIO socket object.

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

Syntax

Connect XLIO socket

```
int xlio_socket_connect(xlio_socket_t sock, const struct sockaddr *to, socklen_t tolen);
```

Where

Argument	Description
sock	XLIO socket object
to	Remote address to connect to
tolen	Length of the address object

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error. Asynchronous connect is a success, therefore, EINPROGRESS and EAGAIN errors are not possible. The result of an asynchronous connect is delivered with the socket event callback. Subsequent xlio_socket_connect() calls are ignored and their return code is undefined.

xlio_socket_setsockopt() and xlio_socket_bind() duplicate setsockopt(2) and bind(2) functionality respectively.

Syntax

setsockopt and bind

```
int xlio_socket_setsockopt(xlio_socket_t sock, int level, int optname, const void *optval, socklen_t optlen);  
int xlio_socket_bind(xlio_socket_t sock, const struct sockaddr *addr, socklen_t addrlen);
```


XLIO exposes protection domain as `ibv_pd` object. The protection domain is related to the outgoing device used by the socket. It is expected to have a protection domain per outgoing interface and, as a result, sockets can share the same object depending on the remote IP address configuration.

`xlio_socket_pd()` should be called after XLIO determines the outgoing device for the socket, which happens in the context of `xlio_socket_connect()`.

The main purpose of the exposed protection domain is to perform memory registration for user's TX data buffers which will be used in the TX zero-copy path. See `ibv_reg_mr(3)` and "TX Data Path" for details. See XLIO Socket sample application for an example.

Syntax

```
Protection domain
```

```
struct ibv_pd *xlio_socket_get_pd(xlio_socket_t sock);
```

Where

Argument	Description
sock	XLIO socket object

Return value

Returns protection domain for the socket on success. On error, NULL is returned. The function fails until `xlio_socket_connect()` is called for the respective socket.

XLIO Socket Events

The socket event callback is configured per polling group with `xlio_poll_group_attr::socket_event_cb()`.

Most of the socket events are delivered from `xlio_poll_group_poll()` context, except for `XLIO_SOCKET_EVENT_TERMINATED`, which can be triggered from the `xlio_socket_destroy()` context.

Socket event callback applies the following restrictions on the socket operations:

- Send operations are allowed only while processing XLIO_SOCKET_EVENT_ESTABLISHED event.
- xlio_socket_destroy() is not allowed.

Send operation in the callback is allowed only for the XLIO_SOCKET_EVENT_ESTABLISHED event.

Syntax

Socket event callback

```
enum {
    /* TCP connection established. */
    XLIO_SOCKET_EVENT_ESTABLISHED = 1,
    /* Socket terminated and no further events are possible. */
    XLIO_SOCKET_EVENT_TERMINATED,
    /* Passive close. */
    XLIO_SOCKET_EVENT_CLOSED,
    /* An error occurred, see the error code value. */
    XLIO_SOCKET_EVENT_ERROR,
};

typedef void (*xlio_socket_event_cb_t)(xlio_socket_t sock, uintptr_t userdata_sq, int event, int value);
```

Where

Argument	Description
sock	XLIO socket object
userdata_sq	Opaque user data which is defined during socket creation
event	Represents the event
value	Holds a POSIX error code for the XLIO_SOCKET_EVENT_ERROR event. Should be ignored for other events

Possible error codes for the `XLIO_SOCKET_EVENT_ERROR` event:

- `ECONNABORTED` - connection aborted by the local side
- `ECONNRESET` - connection reset by the remote side
- `ECONNREFUSED` - connection refused by the remote side during TCP handshake
- `ETIMEDOUT` - connection timed out due to keepalive, user timeout option or TCP handshake timeout

TX Data Path

TX path performs data aggregation until user requests a flush. This allows to avoid data aggregation on the user level and explicitly control sending of more optimal big packets. There are 3 ways to flush sockets:

- Polling group level flush with `xlio_poll_group_flush()`
- Socket level flush with `xlio_socket_flush()`
- Socket level flush with `XLIO_SEND_FLAG_FLUSH` flag in a send operation

It is recommended to use only group level flush for polling groups with `XLIO_GROUP_FLAG_DIRTY` flag. And use socket level flush for sockets from a group without the flag.

Nagle algorithm remains effective for the XLIO sockets, however, it is recommended to use explicit flush mechanism and disable Nagle algorithm with either `TCP_NODELAY` option or `XLIO_TCP_NODELAY` parameter.

By default, send operations are zerocopy. The memory with data must be registered in advance in the XLIO protection domain. See `xlio_socket_get_pd()` and `ibv_reg_mr(3)`.

`XLIO_SEND_FLAG_INLINE` flag forces XLIO to copy data to its internal buffers. An inline send operation does not take ownership on the data memory and the respective buffers may be reused immediately after the call returns. Such an operation ignores `xlio_send_attr::mkey` and `xlio_send_attr::userdata_op` fields.

There is no partial write, and the TCP send buffer option does not affect the XLIO sockets. XLIO either queues all the data or returns an error. Errors are not recoverable.

Flags definitions

```
#define XLIO_SEND_FLAG_FLUSH 0x1
#define XLIO_SEND_FLAG_INLINE 0x2
```

Where

Flag	Description
XLIO_SEND_FLAG_FLUSH	Flush all aggregated data as part of the send operation
XLIO_SENF_FLAG_INLINE	Force XLIO to copy the data to its internal buffers

Types definitions

```
struct xlio_send_attr {
    unsigned flags;
    uint32_t mkey;
    uintptr_t userdata_op;
};
```

Where

Field	Description
flags	Force XLIO to copy the data to its internal buffers
mkey	Memory registration key (e.g. obtained via <code>ibv_reg_mr(3)</code>)

Field	Description
userdata_op	Opaque per-operation userdata

Syntax

Send operation

```
int xlio_socket_send(xlio_socket_t sock, void *data, size_t len, struct xlio_send_attr *attr);
int xlio_socket_sendv(xlio_socket_t sock, struct iovec *iov, unsigned iovcnt, struct xlio_send_attr *attr);
```

Where

Argument	Description
sock	XLIO socket object
data	User pointer to the data to send
len	Length of the data
attr	Send operation attributes
iov	Vectorized data to send
iovcnt	Number of scatter-gather elements in the iov vector

Return value

Returns 0 on success. On error, -1 is returned, and errno is set to indicate the error.

Syntax

Flush operation

```
void xlio_socket_flush(xlio_socket_t sock);  
void xlio_poll_group_flush(xlio_poll_group_t group);
```

Where

Argument	Description
sock	XLIO socket object.
group	XLIO polling group object.

TX Data Path - Zerocopy Completions

User can request a completion on individual zerocopy send operations. A completion is requested with a non-zero `xlio_send_attr::userdata_op` value. Zero value in `xlio_send_attr::userdata_op` disables the completion for the operation. With the completion, XLIO guarantees the following:

- The respective data is delivered to the remote side
- The data is acknowledged by the TCP protocol
- The memory buffer is not used by XLIO

A completion is generated for an operation rather than a buffer. On a completion, user may reuse the respective memory buffers.

XLIO does not guarantee order of the completions. However, completions are likely generated in the same order as their respective send operations.

User may provide duplicate `xlio_send_attr::userdata_op` value in multiple send operations and XLIO generates multiple completions with duplicated `userdata_op` argument respectively.

`XLIO_SEND_FLAG_INLINE` send operations do not generate completions.

Syntax

Zero-copy completion callback

```
void (*socket_comp_cb)(xlio_socket_t sock, uintptr_t userdata_sq, uintptr_t userdata_op);
```

Where

Argument	Description
<code>sock</code>	XLIO socket object.
<code>userdata_sq</code>	Opaque per-socket userdata.
<code>userdata_op</code>	Opaque per-operation userdata.

RX Data Path

RX payload is delivered with the RX callback and treated as an RX event. There is no data aggregation on the socket layer and data is delivered immediately. However, orthogonal features LRO and GRO can perform aggregation on the lower layers, which can affect latency and data granularity.

RX path is always zero-copy - XLIO provides a pointer to its internal buffer, which is in the user address space. Once the RX buffer is handled, the user is responsible to return the buffer back to XLIO.

The user can use an external allocator and/or notification about RX buffers memory allocation to control the memory area, which is used in the RX path. See “Global initialization” section above for details. If needed, the memory area may be prepared in advance for further handling by the application (e.g. register memory for RDMA operations).

XLIO provides an `xlio_buf` metadata object which defines `xlio_buf::userdata`. The field is of uninitialized 8 bytes that can be used by the user during their ownership on the buffer. The user holds ownership on a buffer starting from a respective RX callback and until the buffer is returned back to XLIO.

Syntax

RX data callback

```
void (*socket_rx_cb)(xlio_socket_t sock, uintptr_t userdata_sq, void *data, size_t len, struct xlio_buf *buf);
```

Where

Argument	Description
sock	XLIO socket object
userdata_sq	Opaque per-socket userdata
data	Pointer to the payload which points to an XLIO internal buffer
len	Data length
buf	A buffer metadata object which must be returned back to XLIO

Syntax

Return RX buffer

```
void xlio_socket_buf_free(xlio_socket_t sock, struct xlio_buf *buf);  
void xlio_poll_group_buf_free(xlio_poll_group_t group, struct xlio_buf *buf);
```

Where

Argument	Description
sock	XLIO socket object
group	XLIO polling group object
buf	The metadata object to be returned back to XLIO

Monitoring, Debugging, and Troubleshooting

Monitoring - the xlio_stats Utility

Networking applications open various types of sockets.

The XLIO library holds the following counters:

- Per socket state and performance counters
- Internal performance counters which accumulate information for select(), poll() and epoll_wait() usage by the whole application. An additional performance counter logs the CPU usage of XLIO during select(), poll(), or epoll_wait() calls. XLIO calculates this counter only if XLIO_CPU_USAGE_STATS parameter is enabled, otherwise this counter is not in use and displays the default value as zero.
- XLIO internal CQ performance counters
- XLIO internal RING performance counters
- XLIO internal Buffer Pool performance counters

Use the included xlio_stats utility to view the per-socket information and performance counters during runtime.

Note: For TCP connections, xlio_stats shows only offloaded traffic, and not "os traffic."

Usage:

```
#xlio_stats [-p pid] [-k directory] [-v view] [-d details] [-i interval]
```

The following table lists the basic and additional *xlio_stats* utility options.

Parameter Name	Description	Values
-p, --pid	<pid>	Shows XLIO statistics for a process with pid: <pid>.
-k, --directory	<directory>	Sets shared memory directory path to <directory>

Parameter Name	Description	Values
-n, --name	<application>	Shows XLIO statistics for application: <application>
-f, --find_pid		Finds PID and shows statistics for the XLIO instance running (default).
-F, --forbid_clean		When you set this flag to inactive, shared objects (files) are not removed.
-i, --interval	<n>	Prints a report every <n> seconds. Default: 1 sec
-c, --cycles	<n>	Do <n> report print cycles and exit, use 0 value for infinite. Default: 0
-v, --view	<1 2 3 4 5>	Sets the view type: <ul style="list-style-type: none"> 1. Shows the runtime basic performance counters (default). 2. Shows extra performance counters. 3. Shows additional application runtime configuration information. 4. Shows multicast group membership information. Shows netstat like view of all sockets.
-d, --details	<1 2>	Sets the details mode: <ul style="list-style-type: none"> 1. Show totals (default). Show deltas.
-S, --fd_dump	<fd> [<level>]	Dumps statistics for fd number <fd> using log level <level>. Use 0 value for all open fds.
-z, --zero		Zero counters.

Parameter Name	Description	Values
-l, --log_level	<level>	Sets the XLIO log level to <level> (1 <= level <= 7).
-D, --details_level	<level>	Sets the XLIO log detail level to <level> (0 <= level <= 3).
-s, --sockets	<list range>	Logs only sockets that match <list> or <range> format: 4-16 or 1,9 (or combination).
-V, --version		Prints the version number.
-h, --help		Prints a help message.

Examples

The following sections contain examples of the xlio_stats utility.

Example 1

Description

The following example demonstrates basic use of the xlio_stats utility.

Command Line

```
#xlio_stats -p <pid>
```

⚠ If there is only a single process running over XLIO, it is not necessary to use the `-p` option, since `xlio_stats` will automatically recognize the process.

Output

If no process with a suitable pid is running over the XLIO, the output is:

```
xliostat: Failed to identify process...
```

If an appropriate process was found, the output is:

```
fd      ----- total offloaded -----      ----- total os -----
          pkt  Kbyte  eagain  error  poll%      pkt  Kbyte  error
14 Rx:   140479898 274374      0      0  100.0      0   0      0
      Tx:   140479902 274502      0      0      0      0   0      0
-----
```

Output Analysis

- A single socket with user `fd=14` was created
- Received 140479898 packets, 274374 Kilobytes via the socket
- Transmitted 140479898 packets, 274374 Kilobytes via the socket
- All the traffic was offloaded. No packets were transmitted or received via the OS.
- There were no missed Rx polls (see `XLIO_RX_POLL`). This implies that the receiving thread did not enter a blocked state, and therefore there was no context switch to hurt latency.
- There are no transmission or reception errors on this socket

Example 2

Description

xlio_stats presents not only cumulative statistics, but also enables you to view deltas of XLIO counter updates. This example demonstrates the use of the "deltas" mode.

Command Line

```
#xlio_stats -p <pid> -d 2
```

Output

```
fd      ----- offloaded ----- os -----
      pkt/s Kbyte/s eagain/s error/s poll%  pkt/s Kbyte/s error/s
 15 Rx:   15186    29      0      0    0.0    0      0      0
    Tx:   15186    29      0      0    0.0    0      0      0
 19 Rx:   15186    29      0      0    0.0    0      0      0
    Tx:   15186    29      0      0    0.0    0      0      0
 23 Rx:      0      0      0      0    0.0  15185    22      0
    Tx:      0      0      0      0    0.0  15185    22      0
select() Rx Ready:15185/30372 [os/offload]
Timeouts:0 Errors:0 Poll:100.00% Polling CPU:70%
```

Output Analysis

- Three sockets were created (fds: 15, 19, and 23)
- Received 15186 packets, 29 Kilobytes during the last second via fds: 15 and 19
- Transmitted 15186 packets, 29 Kbytes during the last second via fds: 15 and 19

- Not all the traffic was offloaded, as fd 23: 15185 packets, 22 KBytes were transmitted and received via the OS. This means that fd 23 was used for unicast traffic.
- No transmission or reception errors were detected on any socket
- The application used select for I/O multiplexing
- 45557 packets were placed in socket ready queues (over the course of the last second): 30372 of them offloaded (15186 via fd 15 and 15186 via fd 19), and 15185 were received via the OS (through fd 23)
- There were no missed Select polls (see XLIO_SELECT_POLL). This implies that the receiving thread did not enter a blocked state. Thus, there was no context switch to hurt latency.
- The CPU usage in the select call is 70%
 You can use this information to calculate the division of CPU usage between XLIO and the application. For example when the CPU usage is 100%, 70% is used by XLIO for polling the hardware, and the remaining 30% is used for processing the data by the application.

Example 3

Description

This example presents the most detailed xlio_stats output.

Command Line

```
#xlio_stats -p <pid> -v 3 -d 2
```

Output

```
=====
      Fd= [14]
- Blocked, MC Loop Enabled
- Bound IF  = [0.0.0.0:11111]
- Member of = [224.7.7.7]
```

```

Rx Offload: 1128530 / 786133 / 0 / 0 [kilobytes/packets/eagains/errors]
Rx byte: cur 1470 / max 23520 / dropped/s 0 / limit 16777216
Rx pkt : cur 1 / max 16 / dropped/s 0
Rx poll: 10 / 276077 (100.00%) [miss/hit]
=====
      Fd=[29]
- TCP, Blocked
- Local Address  = [[2003::30]:11112]
- Foreign Address = [[2003::20]:29386]
Tx Offload: 27932 / 2043030 / 0 / 0 [kilobytes/packets/eagains/errors]
Rx Offload: 27932 / 2043030 / 0 / 0 [kilobytes/packets/eagains/errors]
Rx byte: cur 0 / max 14 / dropped 0 / limit 0
Rx pkt : cur 0 / max 1 / dropped 0
Rx RQ Strides: 2043035 / 1 [total/max-per-packet]
Rx poll: 1 / 2043029 (100.00%) [miss/hit]
=====
      CQ=[0]
Packets dropped:          0
Packets queue len:       0
Drained max:             511
Buffer pool size:        500
=====
      RING_ETH=[0]
Rx Offload: 1192953 / 786133 [kilobytes/packets]
Interrupts: 786137 / 78613 [requests/received]
Moderation: 10 / 181 [frame count/usec period]
=====
      BUFFER_POOL(RX)=[0]
Size: 168000
No buffers error: 0
=====
      BUFFER_POOL(TX)=[1]
Size: 199488
No buffers error: 0
=====
      GLOBAL
Pending sockets: 0
Destructed TCP sockets: 4144
Destructed UDP sockets: 0

```


=====

Output Analysis

- A single socket with user fd=14 was created
- The socket is a member of multicast group: 224.7.7.7
- Received 786133 packets, 1128530 Kilobytes via the socket during the last second
- No transmitted data
- All the traffic was offloaded. No packets were transmitted or received via the OS
- There were almost no missed Rx polls (see XLIO_RX_POLL)
- There were no transmission or reception errors on this socket
- The sockets receive buffer size is 16777216 Bytes
- There were no dropped packets caused by the socket receive buffer limit (see XLIO_RX_BYTES_MIN)
- Currently, one packet of 1470 Bytes is located in the socket receive queue
- The maximum number of packets ever located, simultaneously, in the sockets receive queue is 16
- No packets were dropped by the CQ
- No packets in the CQ ready queue (packets which were drained from the CQ and are waiting to be processed by the upper layers)
- The maximum number of packets drained from the CQ during a single drain cycle is 511 (see XLIO_CQ_DRAIN_WCE_MAX)
- The RING_ETH received 786133 packets during this period
- The RING_ETH received 1192953 kilo bytes during this period. This includes headers bytes.
- 786137 interrupts were requested by the ring during this period
- 78613 interrupts were intercepted by the ring during this period
- The moderation engine was set to trigger an interrupt for every 10 packets and with maximum time of 181 usecs
- There were no retransmissions
- The current available buffers in the RX pool is 168000
- The current available buffers in the TX pool is 199488
- There were no buffer requests that failed (no buffer errors)
- There were no sockets pending for destruction
- There were 4144 TCP sockets destructed

Example 4

Description

This example demonstrates how you can get multicast group membership information via xlio_stats.

Command Line

```
#xlio_stats -p <pid> -v 4
```

Output

```
XLIO Group Membership Information
Group          fd number
-----
[224.4.1.3]    15
[224.4.1.2]    19
```

Example 5

Description

This is an example of the “netstat like” view of xlio_stats (-v 5).

Output

Proto	Offloaded	Local Address	Foreign Address	State	Inode	PID
udp	Yes	0.0.0.0:44522	0.0.0.0:*		733679757	1576
tcp	Yes	0.0.0.0:11111	0.0.0.0:*	LISTEN	733679919	1618

Output Analysis

- Two processes are running XLIO
- PID 1576 has one UDP socket bounded to all interfaces on port 44522
- PID 1618 has one TCP listener socket bounded to all interfaces on port 11111

Example 6

Description

This is an example of a log of socket performance counters along with an explanation of the results (using XLIO_STATS_FILE parameter).

Output

```
XLIO: [fd=10] Tx Offload: 455 / 233020 / 0 [kilobytes/packets/errors]
XLIO: [fd=10] Tx OS info: 0 / 0 / 0 [kilobytes/packets/errors]
XLIO: [fd=10] Rx Offload: 455 / 233020 / 0 [kilobytes/packets/errors]
XLIO: [fd=10] Rx OS info: 0 / 0 / 0 [kilobytes/packets/errors]
XLIO: [fd=10] Rx byte: max 200 / dropped 0 (0.00%) / limit 2000000
XLIO: [fd=10] Rx pkt : max 1 / dropped 0 (0.00%)
XLIO: [fd=10] Rx poll: 0 / 233020 (100.00%) [miss/hit]
```

Output Analysis

- No transmission or reception errors occurred on this socket (user fd=10).
- All the traffic was offloaded. No packets were transmitted or received via the OS.
- There were practically no missed Rx polls (see XLIO_RX_POLL and XLIO_SELECT_POLL). This implies that the receiving thread did not enter a blocked state. Thus, there was no context switch to hurt latency.
- There were no dropped packets caused by the socket receive buffer limit (see XLIO_RX_BYTES_MIN). A single socket with user fd=14 was created.

Example 7

Description

This is an example of xlio_stats fd dump utility of established TCP socket using log level = info.

Command Line

```
#xlio_stats --fd_dump 17 info
```

Output

```
XLIO INFO : ===== DUMPING FD 17 STATISTICS =====
XLIO INFO : ===== SOCKET FD =====
XLIO INFO : Fd number : 17
XLIO INFO : Bind info : 22.0.0.4:58795
XLIO INFO : Connection info : 22.0.0.3:6666
XLIO INFO : Protocol : PROTO_TCP
XLIO INFO : Is closed : false
XLIO INFO : Is blocking : true
XLIO INFO : Rx reuse buffer pending : false
```

```

XLIO INFO : Rx reuse buffer postponed : false
XLIO INFO : Is offloaded : true
XLIO INFO : Tx Offload : 12374 / 905105 / 0 / 0 [kilobytes/packets/drops/errors]
XLIO INFO : Rx Offload : 12374 / 905104 / 0 / 0 [kilobytes/packets/eagains/errors]
XLIO INFO : Rx byte : max 14 / dropped 0 (0.00%) / limit 0
XLIO INFO : Rx pkt : max 1 / dropped 0 (0.00%)
XLIO INFO : Rx poll : 0 / 905109 (100.00%) [miss/hit]
XLIO INFO : Socket state : TCP SOCK CONNECTED RDWR
XLIO INFO : Connection state : TCP_CONN_CONNECTED
XLIO INFO : Receive buffer : m_rcvbuff_current 0, m_rcvbuff_max 87380, m_rcvbuff_non_tcp_recved 0
XLIO INFO : Rx lists size : m_rx_pkt_ready_list 0, m_rx_ctl_packets_list 0, m_rx_ctl_reuse_list 0
XLIO INFO : PCB state : ESTABLISHED
XLIO INFO : PCB flags : 0x140
XLIO INFO : Segment size : mss 1460, advtsd_mss 1460
XLIO INFO : Window scaling : ENABLED, rcv_scale 7, snd_scale 7
XLIO INFO : Receive window : rcv_wnd 87380 (682), rcv_ann_wnd 87240 (681), rcv_wnd_max 87380 (682),
rcv_wnd_max_desired 87380 (682)
XLIO INFO : Send window : snd_wnd 87168 (681), snd_wnd_max 8388480 (65535)
XLIO INFO : Congestion : cwnd 1662014
XLIO INFO : Receiver data : rcv_nxt 12678090, rcv_ann_right_edge 12765330
XLIO INFO : Sender data : snd_nxt 12678080, snd_wll 12678076, snd_wl2 12678066
XLIO INFO : Send buffer : snd_buf 255986, max_snd_buff 256000
XLIO INFO : Retransmission : rtime 0, rto 3, nrtx 0
XLIO INFO : RTT variables : rttest 38, rtseq 12678066
XLIO INFO : First unsent : NULL
XLIO INFO : First unacked : seqno 12678066, len 14, seqno + len 12678080
XLIO INFO : Last unacked : seqno 12678066, len 14, seqno + len 12678080
XLIO INFO : Acknowledge : lastack 12678066
XLIO INFO : =====
XLIO INFO : =====

```

Output Analysis

- Fd 17 is a descriptor of established TCP socket (22.0.0.4:58795 -> 22.0.0.3:6666)
- Fd 17 is offloaded by XLIO
- The current usage of the receive buffer is 0 bytes, while the max possible is 87380
- The connection (PCB) flags are TF_WND_SCALE and TF_NODELAY (PCB0x140)
- Window scaling is enabled, receive and send scales equal 7
- Congestion windows equal 1662014

- Unsent queue is empty
- There is a single packet of 14 bytes in the un-acked queue (seqno 12678066)
- The last acknowledge sequence number is 12678066

Example 8

Description

This is an example of `xlio_stats dump=route` utility to dump internal routing caches of an XLIO process.

Command Line

```
#xlio_stats --dump=route
```

Output of xlio_stats

```
xliostat: stats for application: sockperf with pid: 3080863  
xliostat: Dumping Routing information to XLIO using log level = INFO ...
```

Output of XLIO Process

```
XLIO INFO : rtm:142:dump_tbl() Routing table IPv4:  
XLIO INFO : rtm:131:operator()() dst: default gw: 10.210.12.1 dev: eno1 src: 10.210.12.30 table: main scope 0  
type 1 index 2  
XLIO INFO : rtm:131:operator()() dst: 11.4.0.0/16 dev: ens3f0np0 src: 11.4.3.17 table: main scope 253 type 1  
index 8  
XLIO INFO : rtm:131:operator()() dst: 11.4.3.17/32 dev: ens3f0np0 src: 11.4.3.17 table: 255 scope 254 type 2  
index 8
```

```

XLIO INFO : rtm:131:operator()() dst: 127.0.0.1/32 dev: lo src: 127.0.0.1 table: 255 scope 254 type 2 index 1
XLIO INFO : rtm:131:operator()() dst: 127.255.255.255/32 dev: lo src: 127.0.0.1 table: 255 scope 253 type 3
index 1
XLIO INFO : rtm:135:operator()() Total: 5 active and 0 deleted entries.
XLIO INFO : rtm:144:dump_tbl()
XLIO INFO : rtm:145:dump_tbl() Routing table IPv6:
XLIO INFO : rtm:131:operator()() dst: [::1]/128 dev: lo table: main scope 0 type 1 index 1
XLIO INFO : rtm:131:operator()() dst: [2003::]/96 dev: ens3f0np0 table: main scope 0 type 1 index 8
XLIO INFO : rtm:131:operator()() dst: [fd12:3456:789a:1:0:8c0:eb89:8f66]/128 dev: ens3f0np0 table: 255 scope
0 type 2 index 8
XLIO INFO : rtm:131:operator()() dst: [ff15::212:e434:5b0c:7c76]/128 dev: ens3f0np0 table: 255 scope 0 type 1
index 8
XLIO INFO : rtm:131:operator()() dst: [ff00::]/8 dev: ens3f1np1 table: 255 scope 0 type 5 index 9
XLIO INFO : rtm:131:operator()() dst: [ff00::]/8 dev: ens3f1np1 table: 255 scope 0 type 5 index 9
XLIO INFO : rtm:131:operator()() dst: [fe80::]/64 dev: ens3f1np1 table: main scope 0 type 1 index 9
XLIO INFO : rtm:131:operator()() dst: [fe80::d3b7:54fd:5ba8:f8f6]/128 dev: ens3f1np1 table: 255 scope 0 type
2 index 9
XLIO INFO : rtm:135:operator()() Total: 8 active and 1 deleted entries.
XLIO INFO : rtm:148:dump_tbl()
XLIO INFO : rtm:149:dump_tbl() Routing table lookup stats: 31 / 0 [hit/miss]
XLIO INFO : rtm:151:dump_tbl() Routing table update stats: 1511 / 1509 / 0 [new/del/unhandled]

```

Output Analysis

- First section is IPv4 routing
- Second section is IPv6 routing
- First IPv4 entry is default gateway at address 10.210.12.1 with outgoing device eno1
- First IPv6 entry is IPv6 loopback with lo as outgoing device
- Second IPv6 entry is subnet 2003::/96 with outgoing device ens3f0np0
- Second IPv6 entry is from main table

Memory Report

XLIO can print a report in case of insufficient memory for internal buffer pools.

To enable the option set `XLIO_PRINT_REPORT=1`. The report is printed when the application exits normally and XLIO is terminated. Moreover, the report is printed only if insufficient memory was detected.

Sample report:

```
XLIO INFO : XLIO detected insufficient memory. Increasing XLIO_MEMORY_LIMIT can improve performance.
XLIO INFO : Buffer pool 0x1fe2550 (Rx):
XLIO INFO : Buffers: 16 created, 16 free
XLIO INFO : Memory consumption: 128 MB (8 MB per buffer)
XLIO INFO : Requests: 6211 unsatisfied buffer requests
XLIO INFO : Buffer pool 0x1fe2770 (Tx):
XLIO INFO : Buffers: 16384 created, 16384 free
XLIO INFO : Memory consumption: 25 MB (1600 per buffer)
XLIO INFO : Requests: 0 unsatisfied buffer requests
```

Unsatisfied buffer requests: Occurs when buffers are requested from the pool but the pool is unable to fulfil the request due to a memory outage.

Debugging

XLIO Logs

Use the XLIO logs in order to trace XLIO operations. XLIO logs can be controlled by the XLIO_TRACELEVEL variable. This variable's default value is 3, meaning that the only logs obtained are those with severity of PANIC, ERROR, and WARNING.

You can increase the XLIO_TRACELEVEL variable value up to 6 (as described in [XLIO Configuration Parameters](#) to see more information about each thread's operation. Use the XLIO_LOG_DETAILS=3 to add a time stamp to each log line. This can help to check the time difference between different events written to the log.

Use the XLIO_LOG_FILE=/tmp/my_file.log to save the daily events. It is recommended to check these logs for any XLIO WARNINGS and errors. Refer to the [Troubleshooting](#) section to help resolve the different issues in the log.

XLIO will replace a single '%d' appearing in the log file name with the pid of the process loaded with XLIO. This can help in running multiple instances of XLIO each with its own log file name.

When XLIO_LOG_COLORS is enabled, XLIO uses a color scheme when logging: Red for errors and warnings, and dim for low level debugs.

Use the XLIO_HANDLE_SIGSEGV to print a backtrace if a segmentation fault occurs.

Ethernet Counters

Look at the Ethernet counters (by using the `ifconfig` command) to understand whether the traffic is passing through the kernel or through the XLIO (Rx and Tx).

tcpdump

For `tcpdump` to capture offloaded traffic, please follow instructions in section Offloaded Traffic Sniffer in the MLNX_OFED User Manual.

NIC Counters

Look at the NIC counters to monitor HW interface level packets received and sent, drops, errors, and other useful information.

```
ls /sys/class/net/eth2/statistics/
```

Peer Notification Service

Peer notification service handles TCP half-open connections where one side discovers the connection was lost but the other side still see it as active.

The peer-notification daemon is started at system initialization or manually under super user permissions.

The daemon collects information about TCP connections from all the running XLIO processes. Upon XLIO process termination (identified as causing TCP half open connection) the daemon notifies the peers (by sending Reset packets) in order to let them delete the TCP connections on their side.

Troubleshooting

This section lists problems that can occur when using XLIO, and describes solutions for these problems.

1. High Log Level

```
XLIO: WARNING: *****
XLIO: WARNING: *XLIO is currently configured with high log level*
XLIO: WARNING: *Application performance will decrease in this log level!*
XLIO: WARNING: *This log level is recommended for debugging purposes only*
XLIO: WARNING: *****
```

This warning message indicates that you are using XLIO with a high log level.

The XLIO_TRACELEVEL variable value is set to 4 or more, which is good for troubleshooting but not for live runs or performance measurements.

Solution: Set XLIO_TRACELEVEL to its default value 3.

2. On running an application with XLIO, the following error is reported:

```
ERROR: ld.so: object 'libxlio.so' from LD_PRELOAD cannot be preloaded: ignored.
```

Solution: Check that libxlio is properly installed, and that [libxlio.so](#) is located in `/usr/lib` (or in `/usr/lib64`, for 64-bit machines).

3. On attempting to install `libxlio rpm`, the following error is reported:

```
#rpm -ivh libxlio-w.x.y-z.rpm
error: can't create transaction lock
```

Solution: Install the rpm with privileged user (root).

4. The following warning is reported:

```
XLIO: WARNING: *****
XLIO: WARNING: Your current max locked memory is: 33554432. Please change it to unlimited.
XLIO: WARNING: Set this user's default to `ulimit -l unlimited`.
XLIO: WARNING: Read more about this issue in the XLIO's User Manual.
```

```
XLIO: WARNING: *****
```

Solution: When working with root, increase the maximum locked memory to 'unlimited' by using the following command:

```
#ulimit -l unlimited
```

When working as a non-privileged user, ask your administrator to increase the maximum locked memory to unlimited.

5. Lack of huge page resources in the system. The following warning is reported:

```
XLIO WARNING: *****
XLIO WARNING: NO IMMEDIATE ACTION NEEDED!
XLIO WARNING: Not enough suitable hugepages to allocate 2097152 kB.
XLIO WARNING: Allocation will be done with regular pages.
XLIO WARNING: To avoid this message, either increase number of hugepages
XLIO WARNING: or switch to a different memory allocation type:
XLIO WARNING: XLIO_MEM_ALLOC_TYPE=ANON
XLIO INFO : Hugepages info:
XLIO INFO : 1048576 kB : total=0 free=0
XLIO INFO : 2048 kB : total=0 free=0
XLIO WARNING: *****
```

This warning message means that you are using XLIO with huge page memory allocation enabled (XLIO_MEM_ALLOC_TYPE=HUGE), but not enough huge page resources are available in the system. XLIO will use regular pages instead.

If you want XLIO to take full advantage of the performance benefits of huge pages, restart the application after adding more huge page resources to your system or try to free unused huge page shared memory segments with the script below.

```
echo 1000000000 > /proc/sys/kernel/shmmax
echo 800 > /proc/sys/vm/nr_hugepages
```

If you are running multiple instances of your application loaded with XLIO, you will probably need to increase the values used in the above example.

❗ Check that your host machine has enough free memory after allocating the huge page resources for XLIO. Low system memory resources may cause your system to hang.

⚠ Use "ipcs -m" and "ipcrm -m shmid" to check and clean unused shared memory segments.

Use the following script to release XLIO unused huge page resources:

```
for shmid in `ipcs -m | grep 0x00000000 | awk '{print $2}'`;  
do echo 'Clearing' $shmid; ipcrm -m $shmid;  
done;
```

6. Wrong ARP resolution when multiple ports are on the same network.

When two (or more) ports are configured on the same network (e.g. 192.168.1.1/24 and 192.168.1.2/24) XLIO will only detect the MAC address of one of the interfaces. This will result in incorrect ARP resolution.

This is due to the way Linux handles ARP responses in this configuration. By default, Linux returns the same MAC address for both IPs. This behavior is called "ARP Flux".

To fix this, it is required to change some of the interface's settings:

```
$ sysctl -w net.ipv4.conf.[DEVICE].arp_announce=1  
$ sysctl -w net.ipv4.conf.[DEVICE].arp_ignore=2  
$ sysctl -w net.ipv4.conf.[DEVICE].rp_filter=0
```

To verify the issue is resolved, clear the ARP tables on a different server that is on the same network and use the *arping* utility to verify that each IP reports its own MAC address correctly:

```
$ ip -s neigh flush all # clear the arp table on the remote server  
  
$ arping -b -I ens3f1 192.168.1.1  
ARPING 192.168.1.1 from 192.168.1.5 ens3f0  
Unicast reply from 192.168.1.1 [24:8A:07:9A:16:0A] 0.548ms
```

```
$ arping -b -I ens3f1 192.168.1.2
ARPING 192.168.1.2 from 192.168.1.5 ens3f0
Unicast reply from 192.168.1.2 [24:8A:07:9A:16:1A] 0.548ms
```

7. XLIO process cannot establish connection with daemon (xliod) in Microsoft hypervisor environment.

When working with Microsoft Hypervisor, XLIO daemon must be enabled in order to submit Traffic Control (TC) rules which will offload the traffic to the TAP device in case of plug-out events.

The following warning is reported during XLIO startup:

```
XLIO WARNING: *****
XLIO WARNING: * Can not establish connection with the daemon (xliod).      *
XLIO WARNING: * UDP/TCP connections are likely to be limited.              *
XLIO WARNING: *****
```

The following warning is reported during any connection establishment/termination:

```
XLIO WARNING: ring_tap[0x1efc910]:135:attach_flow() Add TC rule failed with error=-19
```

To fix this, run “xliod” as root.

8. Device memory programming is not supported on VMs that lack Blue Flame support.

XLIO will explicitly disable Device Memory capability if it detects Blue Flame support is missing on the node on which user application was launched using XLIO. The following warning message will appear:

```
XLIO WARNING: Device
Memory functionality is not used on devices w/o Blue Flame support.
```

Appendixes

The document contains the following appendixes:

- [Appendix: Sockperf - UDP/TCP Latency and Throughput Benchmarking Tool](#)
- [Appendix: Nginx](#)
- [Appendix: Multicast Routing](#)

Appendix: Sockperf - UDP/TCP Latency and Throughput Benchmarking Tool

This appendix presents *sockperf*, sample application for testing latency and throughput over socket API.

Overview

Sockperf is an open source utility. For more general information, see <https://github.com/Mellanox/sockperf>.

Sockperf's advantage over other network benchmarking utilities is its focus on testing the performance of high-performance systems (as well as testing the performance of regular networking systems). In addition, *sockperf* covers most of the socket API call and options.

Specifically, in addition to the standard throughput tests, *sockperf*:

- Measures latency of each discrete packet at sub-nanosecond resolution (using TSC register that counts CPU ticks with very low overhead).
- Measures latency for ping-pong mode and for latency under load mode. This means that you can measure latency of single packets even under a load of millions of PPS (without waiting for reply of packet before sending a subsequent packet on time).
- Enables spike analysis by providing in each run a histogram with various percentiles of the packets' latencies (for example: median, min, max, 99% percentile, and more) in addition to average and standard deviation.
- Can provide full logs containing all a packet's tx/rx times, without affecting the benchmark itself. The logs can be further analyzed with external tools, such as MS-Excel or matplotlib.
- Supports many optional settings for good coverage of socket API, while still keeping a very low overhead in the fast path to allow cleanest results.

Sockperf operates by sending packets from the client (also known as the *publisher*) to the server (also known as the *consumer*), which then sends all or some of the packets back to the client. This measured roundtrip time is the route trip time (RTT) between the two machines on a specific network path with packets of varying sizes.

- The latency for a given one-way path between the two machines is the RTT divided by two.
- The average RTT is calculated by summing the route trip times for all the packets that perform the round trip and then dividing the total by the number of packets.


Socketperf can test the improvement of UDP/TCP traffic latency when running applications with and without XLIO.

Socketperf can work as a server (*consumer*) or execute under-load, ping-pong, playback and throughput tests as a client (*publisher*).

In addition, socketperf provides more detailed statistical information and analysis, as described in the following section.

Socketperf is installed on the XLIO server at `/usr/bin/socketperf`. For examples of running socketperf, see:

- [Latency with Ping-pong Test](#)
- [Bandwidth and Packet Rate With Throughput Test](#)

 If you want to use multicast, you must first configure the routing table to map multicast addresses to the Ethernet interface, on both client and server. (See [Configuring the Routing Table for Multicast Tests](#)).

Advanced Statistics and Analysis

In each run, socketperf presents additional advanced statistics and analysis information:

- In addition to the average latency and standard deviation, socketperf presents a histogram with various percentiles, including:
 - 50 percentile - the latency value for which 50 percent of the observations are smaller than it. The 50 percentile is also known as the *median*, and is different from the statistical average.
 - 99 percentile - the latency value for which 99 percent of the observations are smaller than it (and 1 percent are higher)

These percentiles, and the other percentiles that the histogram provides, are very useful for analyzing spikes in the network traffic.

- Socketperf can provide a full log of all packets' tx and rx times by dumping all the data that it uses for calculating percentiles and building the histogram to a comma separated file. This file can be further analyzed using external tools such as Microsoft Excel or matplotlib.

All these additional calculations and reports are executed after the fast path is completed. This means that using these options has no effect on the benchmarking of the test itself. During runtime of the fast path, socketperf records txTime and rxTime of packets using the TSC CPU register, which has a negligible effect on the benchmark itself, as opposed to using the computer's clock, which can affect benchmarking results.

Configuring the Routing Table for Multicast Tests

If you wish to use multicast, you must first configure the routing table to map multicast addresses to the Ethernet interface, on both client and server.

Example

```
# route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
# ip -6 route add ff00::/8 dev eth0 metric 1
```

where *eth0* is the Ethernet interface.

You can also set the interface on runtime in sockperf:

- IPv4:
 - Use `--mc-rx-if -<ip>` to set the address of the interface on which to receive multicast packets (can be different from the route table)
 - Use `--mc-tx-if -<ip>` to set the address of the interface on which to transmit multicast packets (can be different from the route table)
- IPv6:
 - Use `--mc-rx-if` to set the interface index on which to receive multicast packets (can be different from the route table)
 - Use `--mc-tx-if` to set the interface index on which to transmit multicast packets (can be different from the route table)

Latency with Ping-pong Test

To measure latency statistics, after the test completes, sockperf calculates the route trip times (divided by two) between the client and the server for all messages, then it provides the average statistics and histogram.

UDP Ping-pong

To run UDP ping-pong:

1. Run the server by using:


```
# sockperf sr -i <server-ip>
```

2. Run the client by using:

```
# sockperf pp -i <server-ip> -m 64
```

Where *-m/--msg-size* is the message size in bytes (minimum default 14).

 For more sockperf Ping-pong options run:

```
# sockperf pp -h
```

TCP Ping-pong

To run TCP ping-pong:

1. Run the server by using:

```
{# sockperf sr -i <server-ip> --tcp
```

2. Run the client by using:

```
# sockperf pp -i <server-ip> --tcp -m 64
```

TCP Ping-pong using XLIO

To run TCP ping-pong using XLIO:

1. Run the server by using:

```
# XLIO_SPEC=latency LD_PRELOAD=libxlio.so sockperf sr -i <server-ip> --tcp
```

2. Run the client by using:

```
# XLIO_SPEC=latency LD_PRELOAD=libxlio.so sockperf pp -i <server-ip> --tcp -m 64
```

Where *xlio_SPEC=latency* is a predefined specification profile for latency.

Bandwidth and Packet Rate with Throughput Test

To determine the maximum bandwidth and highest message rate for a single-process, single-threaded network application, sockperf attempts to send the maximum amount of data in a specific period of time.

UDP MC Throughput

To run UDP MC throughput:

1. On both the client and the server, configure the routing table to map the multicast addresses to the interface by using:

```
# route add -net 224.0.0.0 netmask 240.0.0.0 dev <interface>  
ip -6 route add ff00::/8 dev <interface> metric 1
```

2. Run the server by using:

```
# sockperf sr -i <server-100g-ip>
```


3. Run the client by using:

```
# LD_PRELOAD=libxlio.so sockperf tp -i <server-ip> -m 1472
```

Where *-m/--msg-size* is the message size in bytes (minimum default 14).

4. The following output is obtained:

```
sockperf: Total of 936977 messages sent in 1.100 sec
sockperf: Summary: Message Rate is 851796 [msg/sec]
sockperf: Summary: BandWidth is 1195.759 MBps (9566.068 Mbps)
```

 For more sockperf throughput options run:

```
# sockperf tp -h
```

UDP MC Throughput using XLIO

To run UDP MC throughput:

1. After configuring the routing table as described in [Configuring the Routing Table for Multicast Tests](#), run the server by using:

```
# LD_PRELOAD=libxlio.so sockperf sr -i <server-ip>
```

2. Run the client by using:

```
# LD_PRELOAD=libxlio.so sockperf tp -i <server-ip> -m 1472
```

3. The following output is obtained:

```
sockperf: Total of 4651163 messages sent in 1.100 sec
sockperf: Summary: Message Rate is 4228326 [msg/sec]
sockperf: Summary: BandWidth is 5935.760 MBps (47486.083 Mbps)
```

UDP MC Throughput Summary

Test	100 Gb Ethernet	100 Gb Ethernet + XLIO
Message Rate	851796 [msg/sec]	4228326 [msg/sec]
Bandwidth	1195.759 MBps (9566.068 Mbps)	5935.760 MBps (47486.083 Mbps)
XLIO Improvement	4740.001 MBps (396.4%)	

sockperf Subcommands

You can use additional sockperf subcommands

Usage: sockperf <subcommand> [options] [args]

- To display help for a specific subcommand, use:

```
sockperf <subcommand> --help
```

- To display the program version number, use:

```
sockperf --version
```

Option	Description	For help, use
help (h ,?)	Display a list of supported commands.	
under-load (ul)	Run sockperf client for latency under load test.	# sockperf ul -h
ping-pong (pp)	Run sockperf client for latency test in ping pong mode.	# sockperf pp -h
playback (pb)	Run sockperf client for latency test using playback of predefined traffic, based on timeline and message size.	# sockperf pb -h
throughput (tp)	Run sockperf client for one way throughput test.	# sockperf tp -h
server (sr)	Run sockperf as a server.	# sockperf sr -h

For additional information, see <https://github.com/Mellanox/sockperf>.

Additional Options

The following tables describe additional sockperf options, and their possible values.

Client Options

Short Command	Full Command	Description
-h,-?	--help,--usage	Show the help message and exit.
N/A	--tcp	Use TCP protocol (default UDP).
-i	--ip	Listen on/send to IP <ip>.

Short Command	Full Command	Description
-p	--port	Listen on/connect to port <port> (default 11111).
-f	--file	Read multiple ip+port combinations from file <file> (will use IO muxer '-F').
-F	--iomux-type	Type of multiple file descriptors handle [s select p poll e epoll r recvfrom x socketxtreme](default epoll).
N/A	--timeout	Set select/poll/epoll timeout to <msec> or -1 for infinite (default is 10 msec).
-a	--activity	Measure activity by printing a '.' for the last <N> messages processed.
-A	--Activity	Measure activity by printing the duration for last <N> messages processed.
N/A	--tcp-avoid-nodelay	Stop/Start delivering TCP Messages Immediately (Enable/Disable Nagel). The default is Nagel Disabled except for in Throughput where the default is Nagel enabled.
N/A	--tcp-skip-blocking-send	Enables non-blocking send operation (default OFF).
N/A	--tos	Allows setting tos.
N/A	--mc-rx-if-ip--mc-rx-if	Use mc-rx-ip (IPv4)/mc-rx-if (IPv6). Set ipv4 address/interface index of interface on which to receive multicast messages (can be other than route table).
N/A	--mc-tx-if-ip--mc-tx-if	Use mc-tx-ip (IPv4)/mc-tx-if (IPv6). Set ipv4 address/interface index of interface on which to transmit multicast messages (can be other than route table).
N/A	--mc-loopback-enable	Enable MC loopback (default disabled).
N/A	--mc-ttl	Limit the lifetime of the message (default 2).

Short Command	Full Command	Description
N/A	--mc-source-filter	Set the address <ip, hostname> of the multicast messages source which is allowed to receive from.
N/A	--uc-reuseaddr	Enables unicast reuse address (default disabled).
N/A	--lls	Turn on LLS via socket option (value = usec to poll).
N/A	--buffer-size	Set total socket receive/send buffer <size> in bytes (system defined by default).
N/A	--nonblocked	Open non-blocked sockets.
N/A	--recv_looping_num	Set sockperft to loop over recvfrom() until EAGAIN or <N> good received packets, -1 for infinite, must be used with --nonblocked (default 1).
N/A	--dontwarmup	Do not send warm up packets on start.
N/A	--pre-warmup-wait	Time to wait before sending warm up packets (seconds).
N/A	--vxliozero-copy-read	If possible use XLIO's zero copy reads API (see the XLIOreadme).
N/A	--daemonize	Run as daemon.
N/A	--no-rdtsc	Do not use the register when measuring time; instead use the monotonic clock.
N/A	--load-xlio	Load XLIO dynamically even when LD_PRELOAD was not used.
N/A	--rate-limit	Use rate limit (packet-pacing). When used with XLIO, it must be run with XLIO_RING_ALLOCATION_LOGIC_TX mode.
N/A	--set-sock-accl	Set socket acceleration before running XLIO (available for some NVIDIA systems).

Short Command	Full Command	Description
-d	--debug	Print extra debug information.

Server Options

Short Command	Full Command	Description
N/A	--threads-num	Run <N> threads on server side (requires '-f' option).
N/A	--cpu-affinity	Set threads affinity to the given core IDs in the list format (see: cat /proc/cpuinfo).
N/A	--xliorxfiltercb	If possible use XLIO's receive path packet filter callback API (See the XLIO readme).
N/A	--force-unicast-reply	Force server to reply via unicast.
N/A	--dont-reply	Set server to not reply to the client messages.
-m	--msg-size	Set maximum message size that the server can receive <size> bytes (default 65507).
-g	--gap-detection	Enable gap-detection.

Sending Bursts

Use the "-b (--burst=<size>)" option to control the number of messages sent by the client in every burst.

Debugging sockperf

Use "-d (--debug)" to print extra debug information without affecting the results of the test. The debug information is printed only before or after the fast path.

Troubleshooting sockperf

1. If the following error is received:

```
sockperf error:  
sockperf: No messages were received from the server. Is the server down?
```

Perform troubleshooting as follows:

- Make sure that exactly one server is running
- Check the connection between the client and server
- Check the routing table entries for the multicast/unicast group
- Extend test duration (use the "--time" command line switch)
- If you used extreme values for --mpsand/or --reply-every switch, try other values or try the default values

Appendix: Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. Nginx is free and open-source software, easy to configure in order to serve static web content or to act as a proxy server. Nginx can be deployed to also serve dynamic content on the network, and can serve as a software load balancer. Nginx uses an asynchronous event-driven approach, rather than threads, to handle requests. Nginx's modular event-driven architecture can provide predictable performance under high loads. Official site is <https://www.nginx.com/>

The XLIO library should be configured with --enable-nginx during compilation.

XLIO_NGINX_WORKERS_NUM is used in runtime to provide the same number of workers as Nginx is launched.

Appendix: Multicast Routing

All applications that receive and/or transmit multicast traffic on a multiple-interface host should define the network interfaces through which they would prefer to receive or transmit the various multicast groups.

If a networking application can use existing socket API semantics for multicast packet receive and transmit, the network interface can be defined by mapping the multicast traffic. In this case, the routing table does not have to be updated for multicast group mapping. The socket API `setsockopt` handles these definitions.

When the application uses `setsockopt` with `IP_ADD_MEMBERSHIP/IPv6_JOIN_GROUP` for the receive path multicast join request, it defines the interface through which it wants the XLIO to join the multicast group, and listens for incoming multicast packets for the specified multicast group on the specified socket.

IGMPv3 source specific multicast: when the application uses `setsockopt` with `IP_ADD_SOURCE_MEMBERSHIP/MCAST_JOIN_SOURCE_GROUP` for the receive path multicast join request, it defines the interface through which it wants the XLIO to join the multicast group, and listens for incoming multicast packets for the specified multicast group and from a specified source on the specified socket.

When the application uses `setsockopt` with `IP_MULTICAST_IF/IPv6_MULTICAST_IF` on the transmit path, it defines the interface through which the XLIO will transmit outgoing multicast packets on that specific socket.

If the user application does not use any of the above `setsockopt` socket lib API calls, the XLIO uses the network routing table mapping to find the appropriate interface to be used for receiving or transmitting multicast packets.

Use the `route` command to verify that multicast addresses in the routing table are mapped to the interface you are working on. If they are not mapped, you can map them as follows:

```
#route add -net 224.0.0.0 netmask 240.0.0.0 dev ib0
#ip -6 route add ff00::/8 dev ib0 metric 1
```

It is best to perform the mapping before running the user application with XLIO, so that multicast packets are routed via the 10Gb/s Ethernet interface and not via the default Ethernet interface `eth0`.

The general rule is that the XLIO routing is the same as the OS routing.

Common Abbreviations, Typography and Related Documents

Glossary

Acronym	Definition
API	Application Programmer's Interface
CQ	Completion Queue
FD	File Descriptor
GEth	Gigabit Ethernet Hardware Interface
NIC	Network Interface Card
IB	InfiniBand
IGMP	Internet Group Management Protocol
IP	Internet Protocol
NIC	Network Interface Card
OFED	OpenFabrics Enterprise Distribution
OS	Operating System
pps	Packets Per Second

Acronym	Definition
QP	Queue Pair
RTT	Route Trip Time
UDP	User Datagram Protocol
usec	microseconds
XLIO	Accelerated IO
WCE	Work Completion Elements

Typography

The following table describes typographical conventions used in this document. All terms refer to isolated terms within body text or regular table text unless otherwise mentioned in the Notes column.

Term, Construct, Text Block	Example	Notes
File name, pathname	<i>/opt/ufm/conf/gv.cfg</i>	
Console session (code)	<code>-> flashClear <CR></code>	Complete sample line or block. Comprises both input and output. The code can also be shaded.
Linux shell prompt	<code>#</code>	The "#" character stands for the Linux shell prompt.

Term, Construct, Text Block	Example	Notes
NVIDIA CLI Guest Mode	Switch >	NVIDIA CLI Guest Mode.
NVIDIA CLI admin mode	Switch #	NVIDIA CLI admin mode
String	< > or []	Strings in angled or square brackets are descriptions of what will actually be shown on the screen. For example, the contents of <your-ip> could be 192.168.1.1.
Management GUI label, item name	New Network, New Environment	Management GUI labels and item names appear in bold, whether or not the name is explicitly displayed (for example, buttons and icons).
User text entered into Manager, e.g., to assign as the name of a logical object	"Env1", "Network1"	Note the quotes. The text entered does not include the quotes.

Related Documentation

For additional relevant information, refer to the latest revision of the following documents:

- [Performance Tuning for NVIDIA Adapters](#)
- Performance Tuning Guidelines:
 - For [NVIDIA Network Adapters](#)

User Manual Revision History

Revision	Date	Description
3.31	August 14, 2024	<ul style="list-style-type: none"> Updated XLIO Configuration section. Added XLIO Socket API.
3.30	May 5, 2024	<ul style="list-style-type: none"> Updated version number across examples to reflect the current XLIO version. Updated the following sections: <ul style="list-style-type: none"> Bug Fixes Known Issues Installing XLIO Options XLIO Installation Options
3.20.8	November 31, 2023	Updated MLNX_OFED and Firmware versions to reflect the current XLIO version.
		<ul style="list-style-type: none"> Updated Installing the XLIO Packages Added Building XLIO From Sources and Verifying XLIO Compilation Updated Example under XLIO Configuration Parameters Under Configuration Parameters Values: <ul style="list-style-type: none"> Updated XLIO_RX_WRE_BATCHING parameter Updated XLIO_HW_TS_CONVERSION parameter Removed XLIO_INTERNAL_THREAD_TCP_TIMER_HANDLING parameter Added XLIO_SOCKETXTREME, XLIO_TCP_NODELAY_TRESHOLD, XLIO_DEFERRED_CLOSE, XLIO_TX_SEGS_RING_BATCH_TCP, and XLIO_MULTILOCK parameters
3.0.2	April 30, 2023	Added NVME over TCP DIGEST Offload Tx (Alpha Level) section Substituted actual version numbers with generic examples across the document
2.1.4	February 02, 2023	Updated version number across examples to reflect the current XLIO version
2.0.6	November 02, 2022	Updated version number across examples to reflect the current XLIO version
1.3.5	July 31, 2022	Updated version number across examples to reflect the current XLIO version

Revision	Date	Description
1.2.9	April 28, 2022	<ul style="list-style-type: none"> • Updated examples across the document to reflect the new 1.2.9 XLIO version • Updated several sections to reflect the new support for IPv6 protocol • Updated the example in the third step under Binding XLIO to the Closest NUMA • Updated the example under XLIO Configuration Parameters • Updated the default value for XLIO_HANDLE_SIGINTR and XLIO_HANDLE_SIGSEGV parameters under Configuration Parameters Values • Updated the description for XLIO_TSO parameter under Configuration Parameters Values • Added XLIO_TX_BUFS_BATCH_TCP parameter under Configuration Parameters Values • Updated Supported Ciphers section • Updated the output of example #3 under Monitoring - the xlio_stats Utility
1.1.8	December 5, 2021	<ul style="list-style-type: none"> • Updated examples to reflect the 1.1.8 XLIO version • Updated the following sections <ul style="list-style-type: none"> • Target Applications • TLS HW Offload prerequisites and supported ciphers • Troubleshooting • Deleted the benchmarking example from .Running XLIO v1.1.8.
1.0.6	August 15, 2021	First release

Release Notes Revision History

- [Release Notes Change History](#)
- [Bug Fixes History](#)

Release Notes Change History

Revision	Feature/Category	Description
3.21.2	Socket options	Added SO_XLIO_ISOLATE socket option. See section 7.10
3.20.7	SocketXtreme API	<ul style="list-style-type: none">• Renew support for SocketXtreme API. See section 9.4• Added support for GRO• Added support for TX polling
	Packet Filtering	Renew support for Packet Filtering. See section 9.5
	TCP Acceleration (kernel bypass) for NGINX	<ul style="list-style-type: none">• Added support for NGINX proxy mode.• Renamed XLIO_NGINX_DISTRIBUTE_CQ to XLIO_DISTRIBUTE_CQ
	Memory Allocation	<ul style="list-style-type: none">• Added support for 64K page Kernel• Added support for allocation of different Hugepage sizes• Added memory utilization report• Added XLIO_MEMORY_LIMIT/ XLIO_MEMORY_LIMIT_USER/ XLIO_HUGEPAGE_LOG2/ XLIO_PRINT_REPORT paramters.• Removed 'CONTIG' option from XLIO_MEM_ALLOC_TYPE.
	Ring Migration	Changed default of XLIO_RING_MIGRATION_RATIO_RX to disabled.
	Socket options	Added support for IP_BIND_ADDRESS_NO_PORT socket option.
	Statistics	Added counters for destructed sockets. Added CSV output.

Revision	Feature/Category	Description
	Internal Thread	Added 'delegate' option to XLIO_TCP_CTL_THREAD.
	Bug Fixes	See Bug Fixes section.
3.10.5	Bug Fixes	See Bug Fixes section.
3.0.2	RX Polling Control	Added support for a new parameter to control RX ring polling inside RX API calls - see XLIO_SKIP_POLL_IN_RX under Configuration Parameter Values .
	XLIO Statistics	Added support for Listen socket statistic.
	Performance Improvement	Improved performance of SW GRO (generic-receive-offload).
	NVME over TCP (NVMeoTCP) Offload	[Alpha] Added support for hardware NVMeoTCP DIGEST calculation offload for TX path at alpha level. For further details, please refer to NVME over TCP DIGEST Offload Tx (Alpha Level) section.
	Bug Fixes	See Bug Fixes section.
2.1.4	Multicast Routing	Added support for reading entries and receiving notifications on local routing tables.
	Multipath Routing	Added support for multipath routing through a single hop.
	XLIO Statistics	Enabled dump of routing table through xlio_stats utility.
		Added listen socket statistic.
	Performance Improvement	Improved Tx stability.
	Bug Fixes	See Bug Fixes section.

Revision	Feature/Category	Description
2.0.6	User Space TLS (uTLS) Offload	Added support for uTLS v1.3 offload for Rx path.
	SocketXtreme API	Removed support for SocketXtreme API.
	Packet Filtering	Removed support for Packet Filtering.
	Bug Fixes	See Bug Fixes section.
1.3.5	Connection per Second (CPS) Improvements	Made CPS performance improvements for Hardware TLS offload.
	NGINX Crash Recovery	Added the ability to continue forking new child processes in NGINX application even when a crash takes place. The number of new processes that can be created in a crash scenario is set using XLIO_NGINX_WORKERS_NUM.
1.2.9	IPv6 support	[Beta] Added support for the latest Internet Protocol version IPv6. Note: Support for IPv6 at beta level involves some limitations that can be viewed under Known Issues .
	TLS	Added support for TLS v1.2: 256 bits; TLS v1.3: 128 bits; and TLS v1.3: 256 bits.
	C++11 Standard Support	Product source code is migrated to C++11 standard requirements.
1.1.8	Security	TLS data-path (Tx and Rx) offload allows the NIC to accelerate encryption, decryption, and authentication of AES-GCM (AES128-GCM, TLS v1.2). TLS offload handles data as it goes through the device without storing it, but only updating its context. This results in enhanced host CPU utilization and TLS throughput.
	NGINX with QUIC configuration support	Added QUIC transport protocol, explicitly designed to support multiplexed connections without depending on a single TCP connection.

Revision	Feature/Category	Description
	Performance Improvement	<p>Added Hardware Large Receive Offload (HW LRO) support:</p> <p>Large receive offload (LRO) is a technique for increasing the inbound throughput of high-bandwidth network connections by aggregating multiple incoming TCP packets from a single stream into a larger buffer before they are passed higher up the networking stack.</p> <p>Increased Concurrent Connections maintaining high-wire speed:</p> <p>Supports vertical scaling, limited by available RAM.</p>
1.0.6	TX Flow: Zero Copy	Added support for zero-copy socket send flag MSG_ZEROCOPY with queueing completion notifications on the socket error queue.
	TCP Acceleration (kernel bypass) for NGINX	Added TCP acceleration for NGINX high-performance HTTP server and reverse proxy adaptation.
	uTLS - User Space TLS Offload	TLS hardware offload (Transport Layer Security) is a widely-deployed network protocol used for securing TCP connections on the Internet and accelerating TLS encryption.

Bug Fixes History

The following table describes the issues that have been resolved in previous releases of XLIO.

Internal Ref. Number	Details
3818038	Description: Possible data corruption in small packets when using more than 16 QPs/SQs, in multithreaded environment.
	Keywords: multithreaded, blue flame, data corruption
	Discovered in Version: 3.21.2

Internal Ref. Number	Details
	Fixed in Version: 3.30.5
3704820	<p data-bbox="452 379 2011 432">Description: Create resources for UDP listen socket of HTTP / HTTPS servers in the NGINX configuration file, with the printed error "Failed to create dpcp rq". It should not harm server's ability to handle UDP traffic.</p> <p data-bbox="452 472 943 496">Keywords: Nginx, UDP, QUIC, listen socket, dpcp rq</p> <p data-bbox="452 536 730 560">Discovered in Version: 3.21.2</p> <p data-bbox="452 600 678 624">Fixed in Version: 3.30.5</p>
3574064	<p data-bbox="452 667 2011 719">Description: When transferring large data initially with limited send space, XLIO waits for ACKs to enlarge the Tx window. However, if the queued data does not fit the expanded TCP window, XLIO may not send it. This problem remains until a new send call is made.</p> <p data-bbox="452 759 846 783">Keywords: TCP window, Stuck connection</p> <p data-bbox="452 823 748 847">Discovered in Version: 3.20.1-1</p> <p data-bbox="452 887 678 911">Fixed in Version: 3.21.2</p>
3684889	<p data-bbox="452 954 1429 978">Description: Calling poll() with events=0 on an offloaded TCP socket can return POLLHUP immediately</p> <p data-bbox="452 1018 696 1042">Keywords: poll, POLLHUP</p> <p data-bbox="452 1082 730 1106">Discovered in Version: 3.10.5</p> <p data-bbox="452 1145 678 1169">Fixed in Version: 3.21.2</p>
3612954	Description: An extra retransmission or TCP payload corruption can occur in rare cases after a TCP fast retransmission takes place.

Internal Ref. Number	Details
	<p>Keywords: Fast retransmit</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>
3604029	<p>Description: Library reports error message in multithreaded application using XLIO_RING_MIGRATION_RATIO_RX=-1 XLIO_RING_ALLOCATION_LOGIC_RX=20.</p> <p>Keywords: error, allocation, logic, migration, thread</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>
3602243	<p>Description: XLIO does not operate properly when connect is initialized during link down.</p> <p>Keywords: connect, link, down</p> <p>Discovered in Version: BlueField_OS_4.2.1_OL_8-2.230906.inbox.dev</p> <p>Fixed in Version: 3.20.7</p>
3592264	<p>Description: Fixed XLIO compilation with clang-16 compiler</p> <p>Keywords: clang</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>

Internal Ref. Number	Details
3591210	<p>Description: Fixed TCP stream corruption in case of out of order packets.</p> <p>Keywords: TCP, corruption, out of order</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>
3575217	<p>Description: instead of taking the first IP of the outgoing interface as the source IP of the packet, we go over all the IPs that are set for the outgoing interface and choose the most suitable IP in terms of IP + subnet combination.</p> <p>Keywords: IP, source, packet, subnet</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>
3574901	<p>Description: Closing an outgoing TCP socket might cause a segmentation fault.</p> <p>Keywords: TCP, socket, segmentation, fault</p> <p>Discovered in Version: 3.10</p> <p>Fixed in Version: 3.20.7</p>
3563140	<p>Description: SocketXtream does not currently support Generic Receive Offload (GRO).</p> <p>Keywords: SocketXtraem,GRO, generic receive offload</p> <p>Discovered in Version: 3.10</p>

Internal Ref. Number	Details
	Fixed in Version: 3.20.7
3531871	Description: Fix incorrect socketxtreme completion filling.
	Keywords: socketxtreme
	Discovered in Version: 3.10.3
	Fixed in Version: 3.20.7
3433495	Description: Fixed segfault caused by thread race during RX ring migration
	Keywords: Rx, ring, migration
	Discovered in Version: 3.0.1
	Fixed in Version: 3.20.7
3286324	Description: Fixed RX ring cleanup infinite loop in case of empty RQ.
	Keywords: Rx, ring, cleanup, termination, stuck, hang up
	Discovered in Version: 2.0.7
	Fixed in Version: 3.20.7

Internal Reference Number	Details
3455917	<p>Description: Fixed the issue where a crash took place if an IPv6 multicast group was not previously configured as part of the routing table.</p> <p>Keywords: Multicast; MC; IPv6; routing table</p> <p>Discovered in Version: 3.0.2</p> <p>Fixed in Version: 3.10.5</p>
3447653	<p>Description: Fixed incorrect pbuf chain split in tcp stack in case of re-transmission.</p> <p>Keywords: tcp stack; split</p> <p>Discovered in Version: 2.1.4</p> <p>Fixed in Version: 3.0.2</p>
3226553	<p>Description: Fixed the issue where SO_BINDTODEVICE failed on a connected TCP socket when the interface was the same as the outgoing one.</p> <p>Keywords: SO_BINDTODEVICE; TCP; interface</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.1.4</p>
3263352	<p>Description: Fixed TCP MSS calculation from SYN packet.</p> <p>Keywords: mss; syn</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.1.4</p>

Internal Reference Number	Details
3068125	<p>Description: Added support for Multipath routing.</p> <p>Keywords: ECMP; multipath routing</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.1.4</p>
3213178	<p>Description: Added support for IPv4/IPv6 local routing tables.</p> <p>Keywords: Local routing; IPv4; IPv6</p> <p>Discovered in Version: 2.0.6</p> <p>Fixed in Version: 2.1.4</p>
3100979	<p>Description: When VMA_HANDLE_SIGINTR=0, the following issues are no longer encountered: 1. When SIGINT is caught by VMA, subsequent calls to socket API return EINTR error code immediately. 2. VMA_HANDLE_SIGINTR parameter is ignored by signal() API. However, when VMA_HANDLE_SIGINTR=1, only the first issue persists.</p> <p>Keywords: VMA_HANDLE_SIGINTR; SIGINT; EINTR; signal</p> <p>Discovered in Version: 1.2.9</p> <p>Fixed in Version: 2.0.6</p>

Internal Reference Number	Details
3202977	<p>Description: Fixed the issue of when a segmentation fault took place in RX zero copy API due to buffer trimming int16 overflow.</p> <p>Keywords: Segmentation fault; RX zero copy; buffer</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.0.6</p>
3132299	<p>Description: Fixed the issue of a buffer leak in RX zero copy due to incorrect buffer chain break.</p> <p>Keywords: Buffer leak; RX zero copy</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.0.6</p>
3183546	<p>Description: Fixed the issue where using send* functions with null elements in iov Tx vector caused an API error.</p> <p>Keywords: iov Tx vector</p> <p>Discovered in Version: 1.3.5</p> <p>Fixed in Version: 2.0.6</p>
2678856	<p>Description: Fixed the issue of when attempting to reuse a TCP or UDP port immediately after connection is closed, the application failed with the error "Address already in use".</p> <p>Keywords: Close; bind; EADDRINUSE</p> <p>Discovered in Version: 1.2.9</p>

Internal Reference Number	Details
	Fixed in Version: 1.3.5
3113441	Description: Fixed the issue where resource leakage took place after intensive disconnect/connect attempts.
	Keywords: Resources leakage
	Discovered in Version: 1.2.9
	Fixed in Version: 1.3.5
3113450	Description: Fixed the issue where a crash took place after intensive TCP disconnect/connect attempts.
	Keywords: Crash; tcp
	Discovered in Version: 1.2.9
	Fixed in Version: 1.3.5
3100949	Description: Fixed the issue of when attempting to perform a second connect() after the first connect() has failed, a segmentation fault took place. Now, an expected error is received upon second attempt instead.
	Keywords: connect(); blocking socket; segmentation fault
	Discovered in Version: 1.2.9
	Fixed in Version: 1.3.5

Internal Reference Number	Details
2903631	Description: Fixed a synchronization issue that took place during new TCP socket creation.
	Keywords: TLS
	Discovered in Version: 1.1.8
	Fixed in Version: 1.2.9
2690914	Description: Data corruption during https session.
	Keywords: TLS
	Discovered in Version: N/A
	Fixed in Version: 1.0.6

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. Neither NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make any representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT,

INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of NVIDIA Corporation and/or Mellanox Technologies Ltd. in the U.S. and in other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2024 NVIDIA Corporation & affiliates. All Rights Reserved.

