NVIDIA Messaging Accelerator (VMA) Documentation Rev 9.8.60

Table of contents

Overview

The NVIDIA® Messaging Accelerator (VMA) library accelerates latency-sensitive and throughput-demanding TCP and UDP socket-based applications by offloading traffic from the user-space directly to the network interface card (NIC) or Host Channel Adapter (HCA), without going through the kernel and the standard IP stack (kernel-bypass).

VMA leverages the following benefits:

- Implements the legacy POSIX socket interface
- Increases:
    - Throughput
    - Packets per Second (PPS)
    - Requests per Second (RPS)
- Reduces:
    - Network latency
    - The magnitude of network latency spikes
    - Context switches and interrupts
    - Network congestion
    - Data copying and moving in unicast and multicast applications
- Improves CPU utilization
- Compatible with Ethernet

VMA can work on top of MLNX_OFED driver stack and on a lighter driver stack, MLNX_EN.

Intended Audience

- Market data professionals

- Messaging specialists
- Software engineers and architects
- Systems administrators tasked with installing/uninstalling/maintaining VMA
- ISV partners who want to test/integrate their traffic-consuming/producing applications with VMA

Document Revision History

For the list of changes made to this document, refer to User Manual Revision History.

Release Notes

Revision	Date	Description
9.8.60	November 07, 2024	Added **RH 9.3 (for NVIDIA GH200 only)** as an additional supported OS in System Requirements and Interoperability
	May 06, 2024	Initial release of this document version.

The release note pages provide information on the NVIDIA Messaging Accelerator (VMA), such as changes and new features, system requirements, interoperability parameters, and reports on known software issues as well as 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 VMA version.

Specification	Value
Network Adapter Cards	NVIDIA ConnectX-7 (4x25G)
	NVIDIA ConnectX-7 (200G) Crypto-disabled
	NVIDIA ConnectX-5 / NVIDIA ConnectX-5 Ex

Specification	Value
Firmware	ConnectX-7 28.44.1036
	ConnectX-5 16.35.3502
Driver Stack	• DOCA-Host v2.10.0 • ◦ doca-libvma profile ◦ doca-roce profile **NOTE** : Starting this version, the host driver is part of the NVIDIA DOCA package. For further information, please see NVIDIA MLNX_OFED to DOCA-OFED Transition Guide .
Supported Operating Systems and Kernels	• Ubuntu 20.04 • RedHat 8.6, 8.7, and 9.3 (for NVIDIA GH200 Grace Hopper only)
CPU Architecture	• x86_64 (Intel Xeon) • Arm GH200 Grace Hopper
Minimum memory requirements	1 GB of free memory for installation 800 MB per process running with VMA
Minimum disk space requirements	1 GB
Transport	Ethernet/InfiniBand

VMA Release Contents

	Description
Binary RPM and DEB packages for 64-bit architecture for Linux distribution	• libvma-9.8.60-1.x86_64.rpm • libvma-devel-9.8.60-1.x86_64.rpm • libvma-utils-9.8.60-1.x86_64.rpm • libvma_9.8.60-1_amd64.deb • libvma-dev_9.8.60-1_amd64.deb • libvma-utils_9.8.60-1_amd64.deb

	Description
Documentation	VMA Release Notes VMA Installation and Quick Start Guide VMA User Manual

Certified Applications

The VMA 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

Application	Company / Source	Type	Notes
			x/sockperf)
netperf	Open Source	Bandwidth and Latency Benchmarking	Version 2.6.0
NetPIPE	Open Source	Network Protocol Independent Performance Evaluator	Version 3.7.2
UMS (formerly LBM)	Informatica	Message Middleware Infrastructures	Version 6.7

Changes and New Features

Changes and New Features in this version

Revision	Feature/Category	Description
9.8.60	System Requirements and Interoperability	• Added **Arm GH200 Grace Hopper** as a new supported CPU Architecture • Added DOCA-Host as the new host driver replacing MLNX_OFED
	Bug Fixes	See Bug Fixes in this Version section.

Deprecated Features and Support

- As of VMA v9.0.2, VMA will no longer be backward compatible with MLNX_OFED versions earlier than v5.0-1.0.0.0
- Multi Packet Receive Queue beta functionality is removed as of VMA v9.3.1
- RDMA experimental verbs library (mlnx_lib)
- VMA v9.3.1 and up do not enforce the disable_raw_qp_enforcement option; use the CAP_NET_RAW option instead
- IPoIB is no longer supported with MLNX_OFED v5.1 and above

Important Notes

Note

We recommend using `libnl3` as it is the latest version and includes fixes related to `libnl1`

Note

Bonding Active-Backup (mode 1) is supported with limitations shown in .Known Issues v9.4.0 section.

Bug Fixes in this Version

For the list of older bug fixes, see Bug Fixes History.

Internal Ref. Number	**Details**
3749310	**Description:** Compared to the kernel, the VMA is slower at sending the expected TCP keepAlive probes.
	Keywords: keepalive
	Discovered in Version: 9.8.51
	Fixed in Version: 9.8.60
3749337	**Description:** VMA corresponds differently from the kernel when setting a TCP keepAlive timer to -1
	Keywords: keepalive
	Discovered in Version: 9.8.51
	Fixed in Version: 9.8.60

Known Issues

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

Internal Reference Number	Details
3876349	**Description:** There is a risk of TX data corruption when over 16 application threads are involved in the following conditions: over 16 threads must be handling network sockets, the XLIO_RING_ALLOCATION_LOGIC_TX setting

Internal Reference Number	Details
	equals either 20 or 30, and there is a high traffic rate of packets smaller than 205 bytes.
	Workaround: Either increase the number of BlueFlame registers with MLX5_TOTAL_UUARS env variable or disable BlueFlame doorbell method with MLX5_SHUT_UP_BF=1. Disabling BlueFlame will lead to a latency degradation.
	Keywords: multithreading, high concurrency, data corruption
	Discovered in Version: 9.8.60
3068120	**Description:** Multipath routing is not supported. A warning is printed if such a route is configured in the system. Note that traffic that is routed via a multipath route may be unsent.
	Workaround: Covert multipath routes to a regular route with a single next hop.
	Keywords: ECMP; multipath routing
	Discovered in Version: 9.6.4
2371415	**Description**: In case the switch replicates a UDP packet for both ports, application will get a duplicated packet on a bonding interface.
	Workaround: N/A
	Keywords: RoCE LAG; Bonding
	Discovered in Version: 9.2.2
2394023	**Description:** Active-backup fail_over_mac = 1 mode is not be supported on bonding interface.
	Workaround: N/A
	Keywords: Bonding
	Discovered in Version: 9.2.2
2393571	**Description**: Detaching/attaching slaves on bond interface during runtime is not supported.
	Workaround: N/A
	Keywords: Bonding
	Discovered in Version: 9.2.2

Internal Reference Number	Details
1554637	**Description**: VMA offloads a NetVSC device (Windows Hyper-V network driver) only if SR-IOV is enabled upon application initialization.
	Workaround: N/A
	Keywords: Windows Hypervisor, NetVSC
	Discovered in Version: 8.7.5
1542628	**Description:** Device memory programming is not supported on VMs that lack Blue Flame support.
	Workaround: N/A
	Keywords: MEMIC, Device Memory, Virtual Machine, Blue flame
	Discovered in Version: 8.7.5

1262610	**Description:** When working with Linux guest over Windows Hypervisor, and exceeding the maximum amount of flow steering rules supported by the VM, the following error message will appear:
	`VMA ERROR: rfs[0x32ea410]:273:create_ibv_flow() Create of QP flow ID (tag: 0) failed with flow dst:5.5.5.77:6891, src:0.0.0.0:0, proto:UDP (errno=12 - Cannot allocate memory)`
	`VMA ERROR: ring_simple[0x3292d50]:555:attach_flow() attach_flow=0 failed!`
	As a result, TCP Receive Flow will not be supported, and UDP Receive Flow will not be offloaded.

	Workaround: Reduce the amount of supported VMs for the device in the Hypervisor to increase the total flow steering rules amount for each VM.
	Keywords: Windows Hypervisor, flow steering limit
	Discovered in Version: 8.5.7
1201675	**Description:** When a non-privileged user uses VMA with RHEL inbox to perform networking operations (i.e. allocate IB resources) VMA crashes with a segmentation fault.
	Workaround: Use VMA with root privileges on the RHEL inbox driver.

Internal Reference Number	Details
	Keywords: RHEL inbox driver, segmentation fault
	Discovered in Version: 8.4.10

—	**Description**: The following VMA_ERRORs will be displayed when running ping with root permissions:
	`VMA ERROR: ring_simple[0x7f257d18d720]:256:create_resources() ibv_create_comp_channel for tx failed. m_p_tx_comp_event_channel = (nil) (errno=13 Permission denied)`
	`VMA ERROR: ib_ctx_handler213:mem_dereg() failed de-registering a memory region (errno=13 Permission denied)`

	Workaround: N/A
	Keywords: VMA_ERROR while running ping with root permissions
	Discovered in Version: 8.4.8
965237	**Description**: The following sockets APIs are directed to the OS and are not offloaded by VMA: • int socketpair(int domain, int type, int protocol, int sv[2]); • int dup(int oldfd); • int dup2(int oldfd, int newfd);
	Workaround: N/A
	Keywords: sockets, socketpair, dup, dup2
965227	**Description:** Multicast (MC) loopback within a process is not supported by VMA: • If an application process opens 2 (or more) sockets on the same MC group they will not get each other's traffic.

Internal Reference Number	Details
	Note MC loopback between different VMA processes always work. • Both sockets will receive all ingress traffic coming from the wire
	Workaround: N/A
	Keywords: Multicast, Loopback
965227	**Description**: MC loopback between VMA and the OS limitation. • The OS will reject loopback traffic coming from the NIC • MC traffic from the OS to VMA is functional
	Workaround: N/A
	Keywords: Multicast, Loopback
965227	**Description**: MC loopback Tx is currently disabled and setsockopt (IP_MULTICAST_LOOP) is not supported.
	Workaround: N/A
	Keywords: Multicast, Loopback
919301	**Description**: VMA supports bonding in the following modes: • For active-passive (mode=1), use either fail_over_mac=0 or fail_over_mac=1. • For active-active (mode=4), use fail_over_mac=0. • For VLAN over bond, use fail_over_mac=0 for traffic to be offloaded
	Workaround: N/A
	Keywords: High Availability (HA)
1011005	**Description**: VMA SELECT option supports up to 200 sockets in TCP.
	Workaround: Use ePoll that supports up to 6000 sockets.
	Keywords: SockPerf

Internal Reference Number	Details
977899	**Description**: An unsuccessful trial to connect to a local interface, is reported by VMA as Connection timeout rather that Connection refused.
	Workaround: N/A
	Keywords: Verification
1019085	**Description**: Poll is limited with the amount of sockets.
	Workaround: Use ePoll for large amount of sockets (tested up to 6000)
	Keywords: Poll, ePoll
2394023	**Description**: VLAN on the bond interface does not function properly when bonding is configured with fail_over_mac=1 due to a kernel bug.
	Workaround: Set the fail_over_mac=0
	Keywords: VLAN and High Availability (HA)
—	**Description**: RX UDP UC and MC traffic in Ethernet with fragmented packages (message size is larger than MTU) is not offloaded by VMA and will pass through the Kernel network stack. There might be performance degradation.
	Workaround: N/A
	Keywords: Issues with UDP fragmented traffic reassembly
—	**Description**: The following VMA_PANIC will be displayed when there are not enough open files defined on the server: `VMA PANIC : si[fd=1023]:51:sockinfo() failed to create internal epoll (ret=-1 Too many open files)`
	Workaround: Verify that the number of max open FDs (File Descriptors) in the system (ulimit -n) is twice as number of needed sockets. VMA internal logic requires one additional FD per offloaded socket.
	Keywords: VMA_PANIC while opening large number of sockets
—	**Description**: In MLNX_OFED versions earlier than v4.0-2.0.0.0 and VMA v8.2.10, socket API is only supported in the child process if the parent process has not called any socket routines prior to calling fork. In MLNX_OFED versions 4.0-1.6.1.0, VMA 8.2.10 and later the above restriction no longer exists however the child process cannot use any of

Internal Reference Number	Details
	the parent's socket resources.
	Workaround: VMA supports fork() if VMA_FORK=1 (is enabled) and the Mellanox-supported stack OFED 4.0-1.6.1.0 or later is used. MLNX_OFED support for fork is for kernels supporting the MADV_DONTFORK flag for madvise() (2.6.17 and later), provided that the application does not use threads. The Posix system() call is supported.
	Keywords: There is limited support for fork().
—	**Description**: Applications written in Java use IPv6 by default which is not supported by VMA and may lead to VMA not offloading packets
	Workaround: To change Java to work with IPv4, instruct the application to use "Java -Djava.net.preferIPv4Stack=true"
	Keywords: Java applications using IPv6 stack
—	**Description**: When a VMA-enabled application is running, there are several cases when it does not exit as expected pressing CTRL-C.
	Workaround: Enable SIGINT handling in VMA, by using: `#export VMA_HANDLE_SIGINTR=1`
	Keywords: The VMA application does not exit when you press CTRL-C.
—	**Description**: VMA does not support network interface or route changes during runtime
	Workaround: N/A
	Keywords: Dynamic route or IP changes
—	**Description**: The send rate is higher than the receive rate. Therefore when running one sockperf server with one sockperf client there will be packets loss.
	Workaround: Limit the sender max PPS per receiver capacity. Example below with the following configuration: • O.S: Red Hat Enterprise • Linux Server release 6.2 (Santiago) Kernel \r on an \m • Kernel: 2.6.32-220.el6.x86_64

Internal Reference Number	Details
	• link layer: InfiniBand 56G Etheronet 10G • GEN type: GEN3 • Architecture: x86_64 • CPU: 16 • Core(s) per socket: 8 • CPU socket(s): 2 • NUMA node(s): 2 • Vendor ID: GenuineIntel • CPU family: 6 • Model: 45 • Stepping: 7 • CPU MHz: 2599.926 ``` MC 1 socket max pps 3M MC 10 sockets (select) max pps 1.5M MC 20 sockets (select) max pps 1.5M MC 50 sockets (select) max pps 1M UC 1 socket max pps 2.8M UC 10 sockets max pps 1.5M UC 20 sockets max pps 1.5M UC 50 sockets max pps ```
	Keywords: Packets loss occurs when running sockperf with max pps rate
946914	**Description**: VMA behavior of epoll EPOLLET (Edge Triggered) and EPOLLOUT flags with TCP sockets differs between OS and VMA. • VMA – triggers EPOLLOUT event every received ACK (only data, not syn/fin) • OS – triggers EPOLLOUT event only after buffer was full.
	Workaround: N/A
	Keywords: VMA behavior of epoll EPOLLET (Edge Triggered) and EPOLLOUT flags with TCP sockets
—	**Description**: VMA does not close connections located on the same node (sends FIN to peers) when its own process is terminated.

Internal Reference Number	Details
	Workaround: N/A
	Keywords: VMA connection
—	**Description**: VMA is not closed (sends FIN to peers) when its own process is terminated when the /etc/init.d/vma is stopped.
	Workaround: Launch /etc/init.d/vma start
	Keywords: VMA connection
—	**Description**: When a non-offloaded process joins the same MC address as another VMA process on the same machine, the non-offloaded process does not get any traffic.
	Workaround: Run both processes with VMA
	Keywords: MC traffic with VMA process and non VMA process on the same machine
—	**Description**: Occasionally, epoll with EPOLLONESHOT does not function properly.
	Workaround: N/A
	Keywords: epoll with EPOLLONESHOT

—	**Description**: Occasionally, when running UDP SFNT-STREAM client with poll muxer flag, the client side ends with an expected error:
	`ERROR: Sync messages at end of test lost`
	`ERROR: Test failed.`
	This only occurs with poll flag

	Workaround: Set a higher acknowledgment waiting time value in the sfnt-stream.
	Keywords: SFNT-STREAM UDP with poll muxer flag ends with an error on client side
—	**Description**: Occasionally, SFNT-STREAM UDP client hangs when running multiple times.
	Workaround: Set a higher acknowledgment waiting time value in the sfnt-stream.

Internal Reference Number	Details
	Keywords: SFNT-STREAM UDP client hanging issue
—	**Description**: Ethernet loopback functions only if both sides are either off-loaded or non-offloaded.
	Workaround: N/A
	Keywords: Ethernet loopback is not functional between the VMA and the OS
—	**Description**: The following error may occur when running netperf TCP tests with VMA: `remote error 107 'Transport endpoint is not connected`
	Workaround: Use netperf 2.6.0
	Keywords: Error when running netperf 2.4.4 with VMA
—	**Description**: Occasionally, a packet is not sent if the socket is closed immediately after send() (also for blocking socket)
	Workaround: Wait several seconds after send() before closing the socket
	Keywords: A packet is not sent if the socket is closed immediately after send()
—	**Description**: It can take for VMA more time than the OS to return from an iomux call if all sockets in this iomux are empty sockets
	Workaround: N/A
	Keywords: Iomux call with empty sockets
—	**Description**: TCP throughput with maximum rate may suffer from traffic "hiccups".
	Workaround: Set the mps = 1000000
	Keywords: TCP throughput with maximum rate
—	**Description**: Netcat with VMA on SLES 11 SP1 does not function.
	Workaround: N/A
	Keywords: Netcat on SLES11 SP1

Internal Reference Number	Details
—	**Description**: Sharing of HW resources between the different working threads might cause lock contentions which can affect performance.
	Workaround: Use different ring allocation logics.
	Keywords: Issues with performance with some multi-threaded applications
—	**Description**: Known NetPIPE bug - Netpipe is trying to access read-only memory.
	Workaround: Upgrade to NetPIPE 3.7 or later.
	Keywords: Segmentation fault on NetPIPE exit.
—	**Description**: If VMA runs when VMA_HANDLE_SIGINTR is enabled, an error message might be written upon exiting.
	Workaround: Ignore the error message, or run VMA with VMA_HANDLE_SIGINTR disabled.
	Keywords: When exiting, VMA logs errors when the VMA_HANDLE_SIGINTR is enabled.
—	**Description**: VMA suffers from high latency in low message rates.
	Workaround: Use “Dummy Send”.
	Keywords: VMA ping-pong latency degradation as PPS is lowered
—	**Description**: VMA does not support broadcast traffic.
	Workaround: Use libvma.conf to pass broadcast through OS
	Keywords: No support for direct broadcast
—	**Description**: Directing VMA to access non-valid memory area will cause a segmentation fault.
	Workaround: N/A
	Keywords: There is no non-valid pointer handling in VMA
—	**Description**: VMA allocates resources on the first connect/send operation, which might take up to several tens of milliseconds.
	Workaround: N/A
	Keywords: First connect/send operation might take more time than expected

Internal Reference Number	Details
1452056	**Description**: Calling select upon shutdown of socket will return “ready to write” instead of timeout.
	Workaround: N/A
	Keywords: Calling select() after shutdown (write) returns socket ready to write, while select() is expected to return timeout
875674	**Description**: VMA does not raise sigpipe in connection shutdown.
	Workaround: N/A
	Keywords: VMA does not raise sigpipe
—	**Description**: VMA polls the CQ for packets; if no packets are available in the socket layer, it takes longer.
	Workaround: N/A
	Keywords: When there are no packets in the socket, it takes longer to return from the read call
—	**Description**: Select with more than 1024 sockets is not supported
	Workaround: Compile VMA with SELECT_BIG_SETSIZE defined.
	Keywords: 1024 sockets

Introduction to VMA

VMA Overview

NVIDIA® Messaging Accelerator (VMA) library is a network-traffic offload, dynamically-linked user-space Linux library which serves to transparently enhance the performance of socket-based networking-heavy applications over an Ethernet network. VMA has been designed for latency-sensitive and throughput-demanding, unicast, and multicast applications. VMA can be used to accelerate producer applications and consumer applications and enhance application performance by orders of magnitude without requiring any modification to the application code.

The VMA library accelerates TCP and UDP socket applications, by offloading traffic from the user-space directly to the network interface card (NIC) or Host Channel Adapter (HCA), without going through the kernel and the standard IP stack (kernel-bypass). VMA increases overall traffic packet rate, reduces latency, and improves CPU utilization.

Basic Features

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

- Reduced context switches and interrupts, which result in:
    - Lower latencies
    - Improved CPU utilization
- Minimal buffer copies between user data and hardware – VMA needs only 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 VMA include, but are not limited to:

- Fast transaction-based network applications requiring a high rate of request-response type operations over TCP or UDP unicast, such as a Market Data Order Gateway application working with an exchange.

- Market-data feed-handler software that produces and consumes multicast data feeds, such as Wombat WDF and Reuters RMDS, or any home-grown feed handlers.

- Any other applications that make heavy use of multicast or unicast that require any combination of the following:

    - Higher Packets per Second (PPS) rates than with kernel

    - Lower data distribution latency

    - Lower CPU utilization by the multicast consuming/producing application in order to support further application scalability

Advanced VMA Features

The VMA library provides several significant advantages:

- The underlying wire protocol used for the unicast and multicast solution is standard TCP and UDP IPv4, 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

> **Note**
>
> VMA uses a standard protocol that enables an application to use the VMA for asymmetric acceleration purposes. A “TCP server side” only 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

- Reduced number of context switches. All VMA 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

- Minimal buffer copies. Data is transferred from the hardware (NIC/HCA) straight to the application buffer in user space, with only a single intermediate user space buffer and zero kernel IO buffers
- 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

VMA Library Architecture

Top-Level

The VMA library is a dynamically linked user-space library. Use of the VMA 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 VMA library dynamically without using the LD_PRELOAD parameter, which requires minor application modifications. Using VMA without code modification is described in Running VMA.

When a user application transmits TCP and UDP, unicast and multicast IPv4 data, or listens for such network traffic data, the VMA 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 standard OS network kernel libraries).

VMA implements native RDMA verbs API. The native RDMA verbs have been extended into the Ethernet RDMA-capable 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.

You can implement the code in native RDMA verbs API, without making any changes to your applications. The VMA 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 VMA library operates in a standard networking stack fashion to serve multiple network interfaces.

The VMA 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 HCA or Ethernet NIC, the VMA library intercepts the call and does the bypass work. If the data is passing to/from an unsupported HCA or Ethernet NIC, the VMA library passes the call to the usual kernel libraries responsible for handling network traffic. Thus, the same application can listen in

on multiple HCAs or Ethernet NICs, without requiring any configuration changes for the hybrid environment.

VMA Internal Thread

The VMA library has an internal thread which 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 acknowledgements, retransmissions etc...
- Handle the final closing of TCP sockets
- Update VMA 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 VMA Configuration Parameters for a detailed description.

Socket Types

The following Internet socket types are supported:

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

VMA Installation

Installing VMA

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.

The current version of VMA is compatible with DOCA-host and offers two profile driver stacks: the doca-libvma profile, which supports Ethernet, and the lighter doca-roce profile, which also supports Ethernet.

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

VMA can be installed using one of the following methods (detailed in VMA Installation Options):

- As part of DOCA-host:
    - Installing the combined driver stack and VMA using the **doca-libvma Profile**
    - Installing the driver stack only using the **doca-roce Profile**, then install the VMA library separately using the **libvma Profile**
- Manually:
    - Building VMA From Sources Tree

- Installing VMA via Dedicated Packages

Note

For VMA installation on RHEL 7.x Inbox, see Installing VMA in RHEL 7.x Inbox section.

Note

After completing the installation, verify its success as described in Using sockperf with VMA.

VMA Installation Options

Installing VMA as Part of DOCA-Host

VMAis integrated into the DOCA-Host (doca-libVMA/doca-roce). Therefore, it is recommended to install VMA automatically when installing the NVIDIA DOCA-Host drivers as it depends on drivers' latest firmware, libraries, and kernel modules. This installation assures VMA's correct functionality. Since the DOCA-Host has a plethora of distributions for RHEL, Ubuntu and others, you will have to correctly select the DOCA-Host version that matches your distribution.

This option suits users that wish to install a new VMA version or upgrade to the latest VMA version by overriding the previous one. For further information, see NVIDIA DOCA Installation Guide for Linux section in DOCA documentation.

Installing driver stack & VMA as doca-libvma Profile

For VMA, there is a dedicated DOCA profile called doca-libvma, which installs the driver stack & libvma along with all the necessary dependencies.

Refer to the *NVIDIA DOCA Installation Guide for Linux* section in DOCA documentation and follow the steps for installing doca-all, but use doca-libvma instead. For example:

```
“sudo <your-package-manager> install -y doca-libvma"
```

Installing VMA as libvma Profile

Use the DOCA profile called `libvma`, which includes only the VMA library and its user space dependencies (without driver stack).

For containerized environments, Containers utilize the host system’s kernel, which already provides the required NVIDIA kernel modules and drivers.

Refer to the *NVIDIA DOCA Installation Guide for Linux* section in DOCA documentation and follow the steps for installing doca-all, but use libvma instead. For example:

```
“sudo <your-package-manager> install -y libvma dpcp"
```

Note: libvma-utils, libvma-dev/devel, and sockperf are optional.

- For RPM packages:

  ```
  #sudo <your-package-manager> install -y sockperf libvma-utils
  libvma-devel
  ```

- For DEB packages:

  ```
  #sudo <your-package-manager> install -y sockperf libvma-utils
  libvma-dev
  ```

Installing driver stack as doca-roce Profile

Users looking for a lighter driver stack should install the doca-roce profile and add the libvma-related packages as needed.

Follow the NVIDIA DOCA Installation Guide for Linux section in DOCA documentation to install doca-roce, then add the relevant libvma packages. For example:

```
#sudo  <your-package-manager> install -y doca-roce
```

Installing VMA Manually

Building VMA From Sources

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

Please visit the libvma wiki page for various “how-to” issues.

For building VMA from sources, please visit the following page:
https://github.com/Mellanox/libvma/wiki/Build-Instruction

For adding high debug log level to VMA compile it with:

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

For libvma configuration file please see here.

Verifying VMA Compilation

The compiled VMA library is located under <path-to-vma-sources-root-tree>/src/vma/.libs/libvma.so

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

Example:

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

As indicated in Running VMA, the appearance of the VMA header verifies that the compiled VMA library is loaded with your application.

Remark: it is recommended to keep the original libvma.so under the distribution's original location (e.g. /usr/lib64/libvma.so) and not to override it with the newly complied libvma.so.

Installing VMA via Dedicated Packages

VMA can be installed from a dedicated VMA RPM or a Debian package. In this case, please make sure that NVIDIA's driver stack is already installed and that the driver stack and VMA versions match so that VMA functions correctly.

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

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

VMA includes the following packages which should be saved on your local drive:

- The libvma package contains the binary library shared object file (.so), configuration and documentation files
- The libvma-utils package contains utilities such as vma_stats to monitor vma traffic and statistics
- The libvma-devel (RPM) or libvma-dev (DEB) packages contain VMA's extended API header files, for extra functionality not provided by the socket API

Before you begin, please verify the following prerequisites:

- Check whether VMA is installed:

  For RPM packages, run:

  ```
  #rpm -qil libvma
  ```

  For DEB packages, run:

  ```
  #dpkg –s libvma
  ```

  If the VMA packages are not installed, an appropriate message is displayed.

  If the VMA packages are installed, the RPM or the DEB logs the VMA package information and the installed file list.

- To uninstall the current VMA:

  For RPM packages, run:

  ```
  #rpm -e libvma
  ```

  For DEB packages:

  ```
  #apt-get remove libvma
  ```

Installing the VMA Binary Package

1. Go to the location where the libvma package was saved.
2. Run the command below to start installation:

   • For RPM packages:

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

• For DEB packages:

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

During the installation process:

• VMA library is copied to standard library location (e.g. */usr/lib64/libvma.so*). In addition VMA debug library is copied under the same location (e.g. */usr/lib64/libvma-debug.so*)

• README.txt and version information (VMA_VERSION) are installed at */usr/share/doc/libvma-X.Y.Z/*

• VMA installs its configuration file, *libvma.conf*, to the following location: */etc/libvma.conf*

• The *vmad* service utility is copied into */sbin*

• The script *vma* is installed under */etc/init.d/*. This script can be used to load and unload the VMA service utility.

Installing the VMA 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 libvma-utils-X.Y.Z-R.<arch>.rpm
```

   • For DEB packages:

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

During the installation process the VMA monitoring utility is installed at /usr/bin/vma_stats.

Installing the VMA 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 libvma-devel-X.Y.Z-R.<arch>.rpm
   ```

   • For DEB packages:

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

   During the installation process the VMA extra header file is installed at /usr/include/mellanox/vma_extra.h.

Installing VMA in RHEL 7.x Inbox

Note

VMA can be installed on RHEL 7.2 and above Inbox Drivers.

Note

RHEL 7.x distribution contains RoCE drivers version which can be retrieved by running “yum install”. The NVIDIA drivers contain a code that was tested and integrated into the Linux upstream. However, this code may be more or less advanced than MLNX_OFED version 4.0-x.x.x or MLNX-EN v3.4-x.x.x.

This option is suitable for users who do not wish to change their exiting NVIDIA driver that comes with RHEL 7.x installation.

RHEL 7.2 with Inbox drivers, please see here.

For running VMA over RHEL 7.3 or later with Inbox drivers, execute:

```
yum install libvma
```

For libvma configuration file please visit here.

For information on how to uninstall VMA in RHEL 7.x Inbox, please refer to VMA in RHEL 7.x Inbox Uninstallation.

Note

By default “yum install libvma” will install one of past VMA versions. In order to run with latest VMA version, you can compile libvma directly from GitHub.. For further information, refer to Verifying VMA Compilation.

Note

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

Running VMA

This section shows how to run a simple network benchmarking test and compare the kernel network stack results to VMA.

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

Example:

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

Note

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

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

```
VMA INFO: VMA_VERSION: X.Y.Z-R Release built on MM DD YYYY
HH:mm:ss
VMA INFO: Cmd Line: sockperf server -i 11.4.3.3
VMA INFO: OFED Version: MLNX_OFED_LINUX-X.X-X.X.X.X:
```

```
VMA INFO: ---------------------------------------------------------
--------------------
```

The output will always show:

- The VMA version
- The application's name (in the above example: Cmd Line: sockperf sr)

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

Running VMA using non-root Permission

1. Check if the LD can find the libvma library.

```
ld -lvma –verbose
```

2. Set the UID bit to enforce user ownership.

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

3. Grant `CAP_NET_RAW` privileges to the application.

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

4. Launch the application under no root.

```
LD_PRELOAD=libvma.so sockperf sr --tcp -i 10.0.0.4 -p 12345
```

```
LD_PRELOAD=libvma.so sockperf pp --tcp -i 10.0.0.4 -p 12345 -t10
```

Benchmarking Example

Prerequisites

- Install sockperf – a tool for network performance measurement

  This can be done by either

  - Downloading and building from source from: https://github.com/Mellanox/sockperf
  - Using

    ```
    yum install: yum install sockperf
    ```

- Two machines, one serves as the server and the second as a client
  - Management interfaces configured with an IP that machines can ping each other
  - Physical installation of an NVIDIA® NIC in your machines
- Your system must recognize the NVIDIA® NIC. To verify it recognizes it, run:

  ```
  lspci | grep Mellanox
  ```

  Output example:

  ```
  $ lspci | grep Mellanox
  82:00.0 Ethernet controller: Mellanox Technologies MT28800
  Family [ConnectX-5 Ex]
  ```

```
82:00.1 Ethernet controller: Mellanox Technologies MT28800
Family [ConnectX-5 Ex]
```

Kernel Performance

Kernel Performance Server Side

On the first machine run:

```
$ sockperf server -i 11.4.3.3
```

Server side example output:

```
sockperf: [SERVER] listen on:sockperf: == version #3.7-no.git ==
sockperf: [SERVER] listen on:
[ 0] IP = 11.4.3.3         PORT = 11111 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: [tid 124545] using recvfrom() to block on socket(s)
```

Kernel Performance Client Side

On the second machine run:

```
$ sockperf ping-pong -t 4 -i 11.4.3.3
```

Client-side example output:

```
sockperf: == version #3.7-no.git ==
```

```
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on
socket(s)

[ 0] IP = 11.4.3.3         PORT = 11111 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: Starting test...
sockperf: Test end (interrupted by timer)
sockperf: Test ended
sockperf: [Total Run] RunTime=4.000 sec; Warm up time=400 msec;
SentMessages=307425; ReceivedMessages=307424
sockperf: ========== Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=3.550 sec; SentMessages=272899;
ReceivedMessages=272899
sockperf: ====> avg-lat=  6.488 (std-dev=0.396)
sockperf: # dropped messages = 0; # duplicated messages = 0; #
out-of-order messages = 0
sockperf: Summary: Latency is 6.488 usec
sockperf: Total 272899 observations; each percentile contains
2728.99 observations
sockperf: ---> <MAX> observation =    20.484
sockperf: ---> percentile 99.999 =    17.732
sockperf: ---> percentile 99.990 =     9.364
sockperf: ---> percentile 99.900 =     8.491
sockperf: ---> percentile 99.000 =     7.963
sockperf: ---> percentile 90.000 =     6.975
sockperf: ---> percentile 75.000 =     6.831
sockperf: ---> percentile 50.000 =     6.307
sockperf: ---> percentile 25.000 =     6.212
sockperf: ---> <MIN> observation =     5.887
```

VMA Latency

Check the VMA performance by running sockperf and using the "VMA_SPEC=latency" environment variable.

VMA Performance Server Side

On the first machine run:

```
$ LD_PRELOAD=libvma.so VMA_SPEC=latency sockperf server -i 11.4.3.3
```

Server-side example output:

```
VMA INFO: VMA_VERSION: X.Y.Z-R Release built on MM DD YYYY
HH:mm:ss
VMA INFO: Cmd Line: sockperf server -i 11.4.3.3
VMA INFO: OFED Version: MLNX_OFED_LINUX-X.X-X.X.X.X:
VMA INFO: ----------------------------------------------------------
--------------------
VMA INFO: VMA Spec                          Latency
[VMA_SPEC]
VMA INFO: Log Level                         INFO
[VMA_TRACELEVEL]
VMA INFO: Ring On Device Memory TX          16384
[VMA_RING_DEV_MEM_TX]
VMA INFO: Tx QP WRE                         256
[VMA_TX_WRE]
VMA INFO: Tx QP WRE Batching                4
[VMA_TX_WRE_BATCHING]
VMA INFO: Rx QP WRE                         256
[VMA_RX_WRE]
VMA INFO: Rx QP WRE Batching                4
[VMA_RX_WRE_BATCHING]
VMA INFO: Rx Poll Loops                     -1
[VMA_RX_POLL]
VMA INFO: Rx Prefetch Bytes Before Poll     256
[VMA_RX_PREFETCH_BYTES_BEFORE_POLL]
VMA INFO: GRO max streams                   0
[VMA_GRO_STREAMS_MAX]
```

```
VMA INFO: Select Poll (usec)            -1
[VMA_SELECT_POLL]
VMA INFO: Select Poll OS Force          Enabled
[VMA_SELECT_POLL_OS_FORCE]
VMA INFO: Select Poll OS Ratio          1
[VMA_SELECT_POLL_OS_RATIO]
VMA INFO: Select Skip OS                1
[VMA_SELECT_SKIP_OS]
VMA INFO: CQ Drain Interval (msec)      100
[VMA_PROGRESS_ENGINE_INTERVAL]
VMA INFO: CQ Interrupts Moderation      Disabled
[VMA_CQ_MODERATION_ENABLE]
VMA INFO: CQ AIM Max Count              128
[VMA_CQ_AIM_MAX_COUNT]
VMA INFO: CQ Adaptive Moderation        Disabled
[VMA_CQ_AIM_INTERVAL_MSEC]
VMA INFO: CQ Keeps QP Full              Disabled
[VMA_CQ_KEEP_QP_FULL]
VMA INFO: TCP nodelay                   1
[VMA_TCP_NODELAY]
VMA INFO: Avoid sys-calls on tcp fd     Enabled
[VMA_AVOID_SYS_CALLS_ON_TCP_FD]
VMA INFO: Internal Thread Affinity      0
[VMA_INTERNAL_THREAD_AFFINITY]
VMA INFO: Thread mode                   Single
[VMA_THREAD_MODE]
VMA INFO: Mem Allocate type             2 (Huge Pages)
[VMA_MEM_ALLOC_TYPE]
VMA INFO: ------------------------------------------------------------
--------------------
sockperf: == version #3.7-no.git ==
sockperf: [SERVER] listen on:
[ 0] IP = 11.4.3.3         PORT = 11111 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: [tid 124588] using recvfrom() to block on socket(s)
```

VMA Performance Client Side

On the second machine run:

```
$ LD_PRELOAD=libvma.so VMA_SPEC=latency sockperf ping-pong -t 4 -
i 11.4.3.3
```

Client-side example output:

```
VMA INFO: VMA_VERSION: X.Y.Z-R Release built on MM DD YYYY
HH:mm:ss
VMA INFO: Cmd Line: sockperf server -i 11.4.3.3
VMA INFO: OFED Version: MLNX_OFED_LINUX-X.X-X.X.X.X:
VMA INFO: ----------------------------------------------------------
--------------------
VMA INFO: VMA Spec                          Latency
[VMA_SPEC]
VMA INFO: Log Level                         INFO
[VMA_TRACELEVEL]
VMA INFO: Ring On Device Memory TX          16384
[VMA_RING_DEV_MEM_TX]
VMA INFO: Tx QP WRE                         256
[VMA_TX_WRE]
VMA INFO: Tx QP WRE Batching                4
[VMA_TX_WRE_BATCHING]
VMA INFO: Rx QP WRE                         256
[VMA_RX_WRE]
VMA INFO: Rx QP WRE Batching                4
[VMA_RX_WRE_BATCHING]
VMA INFO: Rx Poll Loops                     -1
[VMA_RX_POLL]
```

```
VMA INFO: Rx Prefetch Bytes Before Poll  256
[VMA_RX_PREFETCH_BYTES_BEFORE_POLL]
VMA INFO: GRO max streams                0
[VMA_GRO_STREAMS_MAX]
VMA INFO: Select Poll (usec)             -1
[VMA_SELECT_POLL]
VMA INFO: Select Poll OS Force           Enabled
[VMA_SELECT_POLL_OS_FORCE]
VMA INFO: Select Poll OS Ratio           1
[VMA_SELECT_POLL_OS_RATIO]
VMA INFO: Select Skip OS                 1
[VMA_SELECT_SKIP_OS]
VMA INFO: CQ Drain Interval (msec)       100
[VMA_PROGRESS_ENGINE_INTERVAL]
VMA INFO: CQ Interrupts Moderation       Disabled
[VMA_CQ_MODERATION_ENABLE]
VMA INFO: CQ AIM Max Count               128
[VMA_CQ_AIM_MAX_COUNT]
VMA INFO: CQ Adaptive Moderation         Disabled
[VMA_CQ_AIM_INTERVAL_MSEC]
VMA INFO: CQ Keeps QP Full               Disabled
[VMA_CQ_KEEP_QP_FULL]
VMA INFO: TCP nodelay                    1
[VMA_TCP_NODELAY]
VMA INFO: Avoid sys-calls on tcp fd      Enabled
[VMA_AVOID_SYS_CALLS_ON_TCP_FD]
VMA INFO: Internal Thread Affinity       0
[VMA_INTERNAL_THREAD_AFFINITY]
VMA INFO: Thread mode                    Single
[VMA_THREAD_MODE]
VMA INFO: Mem Allocate type              2 (Huge Pages)
[VMA_MEM_ALLOC_TYPE]
VMA INFO: ------------------------------------------------------------
--------------------
sockperf: == version #3.7-no.git ==
```

```
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on
socket(s)

[ 0] IP = 11.4.3.3         PORT = 11111 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: Starting test...
sockperf: Test end (interrupted by timer)
sockperf: Test ended
sockperf: [Total Run] RunTime=4.000 sec; Warm up time=400 msec;
SentMessages=1855851; ReceivedMessages=1855850
sockperf: ========== Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=3.550 sec; SentMessages=1656957;
ReceivedMessages=1656957
sockperf: ====> avg-lat=  1.056 (std-dev=0.074)
sockperf: # dropped messages = 0; # duplicated messages = 0; #
out-of-order messages = 0
sockperf: Summary: Latency is 1.056 usec
sockperf: Total 1656957 observations; each percentile contains
16569.57 observations
sockperf: ---> <MAX> observation =    4.176
sockperf: ---> percentile 99.999 =    1.639
sockperf: ---> percentile 99.990 =    1.552
sockperf: ---> percentile 99.900 =    1.497
sockperf: ---> percentile 99.000 =    1.305
sockperf: ---> percentile 90.000 =    1.179
sockperf: ---> percentile 75.000 =    1.054
sockperf: ---> percentile 50.000 =    1.031
sockperf: ---> percentile 25.000 =    1.015
sockperf: ---> <MIN> observation =    0.954
```

Comparing Results

VMA is showing over 614.3% performance improvement comparing to kernel

Average latency:

- Using Kernel 6.488 usec
- Using VMA 1.056 usec

Percentile latencies:

Percentile	Kernel	VMA
Max	20.484	4.176
99.999	17.732	1.639
99.990	9.364	1.552
99.900	8.491	1.497
99.000	7.963	1.305
90.000	6.975	1.179
75.000	6.831	1.054
50.000	6.307	1.031
25.000	6.212	1.015
MIN	5.887	0.954

In order to tune your system and get best performance see section Basic Performance Tuning.

Libvma-debug.so

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

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

Example:

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

Note

libvma-debug.so is located in the same library path as libvma.so under your distribution's OS.

For example in RHEL7.x x86_64, the libvma.so is located in /usr/lib64/libvma-debug.so.

Note

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

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

See section Building VMA from Sources.

Uninstalling VMA

Automatic VMA Uninstallation

If you are about to install a new NVIDIA® driver version, the old VMA version will be automatically uninstalled as part of this process (followed by a new VMA version

installation). Please refer to Installing VMA Binary as Part of NVIDIA Drivers for installing NVIDIA® drivers command details.

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

Manual VMA Uninstallation

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

- For RPM packages:

```
#rpm -e libvma-utils
#rpm -e libvma-devel
#rpm -e libvma
```

- For DEB packages:

```
#dpkg -r libvma-utils
#dpkg -r libvma-dev
#dpkg -r libvma
```

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

VMA in RHEL 7.x Inbox Uninstallation

VMA under RHEL 7.x is built from sources and is not installed as package. Therefore if VMA is not needed, delete the compiled libvma.so (or the entire libvma sources) and the configuration file (e.g. /etc/libvma.conf).

Note: Uninstalling RoCE drivers' package using "yum remove" will neither delete the compiled VMA library nor the configuration file.

To uninstall VMA, run:

```
rpm –e libvma
```

Upgrading libvma.conf

When you upgrade VMA (through automatic or manual installation), the *libvma.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

Port Type Management / VPI Cards Configuration

NVIDIA ConnectX ports can be individually configured to work as Ethernet 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, please refer to MFT documentation.

Basic Performance Tuning

Please see the Tuning Guide and VMA Performance Tuning Guide for detailed instructions on how to optimally tune your machines for VMA performance.

VMA Tuning Parameters

Parameter	Description	Example
VMA_SPEC	Optimized performance can easily be measured by VMA predefined specification profile for latency: • Latency profile spec – optimized latency on all use cases. System is tuned to keep balance between Kernel and VMA. **Note** It may limit the maximum bandwidth.	`LD_PRELOAD=libvma.so VMA_SPEC=latency sockperf ping-pong -t 10 -i 11.4.3.1`

VMA_RX_POLL	For blocking sockets only. It controls the number of times the ready packets can be polled on the RX path before they go to sleep (wait for interrupt in blocked mode). The recommended value for best latency is -1 (unlimited).
	• For best latency, use -1 for infinite polling
	• For low CPU usage use 1 for single poll
	• Default value is 100000
		• Server:
		`VMA_RX_POLL=-1 LD_PRELOAD=libvma.so taskset -c 15 sockperf sr -i 17.209.13.142`
		• Client:
		`VMA_RX_POLL=-1 LD_PRELOAD=libvma.so taskset -c 15 sockperf pp -i 17.209.13.142 -t 5`

VMA_INTERNAL_T	Controls which CPU core(s) the VMA internal thread is serviced on. The recommended configuration is to run VMA internal thread on a different	• Server:

Parameter	Description	Example

HREAD_AFFINITY	core than the application but on the same NUMA node.	`VMA_INTERNAL_THREAD_AFFINITY=14 VMA_RX_POLL=-1 LD_PRELOAD=libvma.so taskset -c 15 sockperf sr -i 17.209.13.142`
		• Client:
		`VMA_INTERNAL_THREAD_AFFINITY= 14 VMA_RX_POLL=-1 LD_PRELOAD=libvma.so taskset -c 15 sockperf pp -i 17.209.13.142 -t 5`

Binding VMA 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 1.

2. Check which CPU is related to the specific NUMA.

```
[root@r-host144 ~]# lscpu
NUMA node0 CPU(s):     0-13,28-41
NUMA node1 CPU(s):     14-27,42-55
```

The output above shows that:

• CPUs 0-13 & 28-41 are related to NUMA 0 • CPUs 14-27 & 42-55 are related to NUMA 1

Since 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 VMA process on a specific CPU.

• Server side:

```
LD_PRELOAD=libvma.so taskset -c 15 sockperf sr -i < MLX IP
interface >
```

• Client side:

```
LD_PRELOAD=libvma.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

Note

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 better

   If all of your system jitter is under control, it is recommended to turn is 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.

VMA Configuration

You can control the behavior of VMA by configuring:

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

Configuring libvma.conf

The installation process creates a default configuration file, */etc/libvma.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, VMA control settings apply to all applications.
- The transport protocol to be used for the created sockets.
- The IP addresses and ports in which you want to offload.

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

In the *libvma.conf* file:

- You can define different VMA 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 *libvma.conf*. For a sample *libvma.conf* file, see Example of VMA Configuration.

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: • db2* matches any program with a name starting with db2. • t?cp matches ttcp, etc.
<user-defined-id|*>	Specify the process ID to which the control statements appearing below this statement apply. **Note** You must also set the VMA_APPLICATION_ID environment variable to the same value as *user-defined-id*.

Configuring Socket Transport Control

Use socket control statements to specify when libvma will offload AF_INET/SOCK_STREAM or AF_INET/SOCK_DATAGRAM 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: • vma – VMA 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 *vma*.
role	Specify one of the following roles: • 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>]|*

Option	Description
	• 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: `start-port[-end-port]` Port range: 0-65536

Example of VMA Configuration

To set the following:

- Apply the rules to program *tcp_lat* with ID *B1*
- Use VMA 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 *libvma.conf*, configure:

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

Note

You must also set the VMA parameter:

```
VMA_APPLICATION_ID=B1
```

VMA Configuration Parameters

VMA configuration parameters are Linux OS environment variables that are controlled with system environment variables.

It is recommended that you set these parameters prior to loading the application with VMA. 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 VMA library prints the VMA version information, as well as the configuration parameters being used and their values to stderr.

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

- VMA_TRACELEVEL
- VMA_LOG_FILE

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

> **Note**
>
> The VMA version information, parameters, and values are subject to change.

For example:

```
VMA INFO: VMA_VERSION: X.Y.Z-R Release built on MM DD YYYY
HH:mm:ss
VMA INFO: Cmd Line: sockperf server -i 11.4.3.3
VMA INFO: OFED Version: MLNX_OFED_LINUX-X.X-X.X.X.X:
VMA INFO: ----------------------------------------------------------
--------------------
 Pid:  2378 Tid:  2378 VMA INFO: Log Level
DEBUG                          [VMA_TRACELEVEL]
 Pid:  2378 Tid:  2378 VMA INFO: Log Details
2                              [VMA_LOG_DETAILS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Log Colors
Enabled                        [VMA_LOG_COLORS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Log File
[VMA_LOG_FILE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Stats File
[VMA_STATS_FILE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Stats shared memory directory
/tmp/                          [VMA_STATS_SHMEM_DIR]
 Pid:  2378 Tid:  2378 VMA DETAILS: VMAD output directory
/tmp/vma                       [VMA_VMAD_NOTIFY_DIR]
 Pid:  2378 Tid:  2378 VMA DETAILS: Stats FD Num (max)
100                           [VMA_STATS_FD_NUM]
 Pid:  2378 Tid:  2378 VMA DETAILS: Conf File
/etc/libvma.conf               [VMA_CONFIG_FILE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Application ID
VMA_DEFAULT_APPLICATION_ID [VMA_APPLICATION_ID]
 Pid:  2378 Tid:  2378 VMA DETAILS: Polling CPU idle usage
Disabled                       [VMA_CPU_USAGE_STATS]
 Pid:  2378 Tid:  2378 VMA DETAILS: SigIntr Ctrl-C Handle
Disabled                       [VMA_HANDLE_SIGINTR]
 Pid:  2378 Tid:  2378 VMA DETAILS: SegFault Backtrace
Disabled                       [VMA_HANDLE_SIGSEGV]
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring allocation logic TX
0 (Ring per interface)     [VMA_RING_ALLOCATION_LOGIC_TX]
```

```
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring allocation logic RX
0 (Ring per interface)      [VMA_RING_ALLOCATION_LOGIC_RX]
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring migration ratio TX
100                          [VMA_RING_MIGRATION_RATIO_TX]
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring migration ratio RX
100                          [VMA_RING_MIGRATION_RATIO_RX]
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring limit per interface        0
(no limit)                  [VMA_RING_LIMIT_PER_INTERFACE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Ring On Device Memory TX
0                            [VMA_RING_DEV_MEM_TX]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP max syn rate
0 (no limit)                 [VMA_TCP_MAX_SYN_RATE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx Mem Segs TCP
1000000                     [VMA_TX_SEGS_TCP]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx Mem Bufs
200000                      [VMA_TX_BUFS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx QP WRE
2048                        [VMA_TX_WRE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx QP WRE Batching
64                           [VMA_TX_WRE_BATCHING]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx Max QP INLINE
204                          [VMA_TX_MAX_INLINE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx MC Loopback
Enabled                      [VMA_TX_MC_LOOPBACK]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx non-blocked eagains
Disabled                     [VMA_TX_NONBLOCKED_EAGAINS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Tx Prefetch Bytes
256                          [VMA_TX_PREFETCH_BYTES]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Mem Bufs
200000                      [VMA_RX_BUFS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx QP WRE
16000                       [VMA_RX_WRE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx QP WRE Batching
64                           [VMA_RX_WRE_BATCHING]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Byte Min Limit
65536                       [VMA_RX_BYTES_MIN]
```

```
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Poll Loops
100000                        [VMA_RX_POLL]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Poll Init Loops
0                             [VMA_RX_POLL_INIT]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx UDP Poll OS Ratio
100                           [VMA_RX_UDP_POLL_OS_RATIO]
 Pid:  2378 Tid:  2378 VMA DETAILS: HW TS Conversion
3                             [VMA_HW_TS_CONVERSION]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Poll Yield
Disabled                      [VMA_RX_POLL_YIELD]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Prefetch Bytes
256                           [VMA_RX_PREFETCH_BYTES]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx Prefetch Bytes Before Poll
0                             [VMA_RX_PREFETCH_BYTES_BEFORE_POLL]
 Pid:  2378 Tid:  2378 VMA DETAILS: Rx CQ Drain Rate
Disabled                      [VMA_RX_CQ_DRAIN_RATE_NSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: GRO max streams
32                            [VMA_GRO_STREAMS_MAX]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP 3T rules
Disabled                      [VMA_TCP_3T_RULES]
 Pid:  2378 Tid:  2378 VMA DETAILS: UDP 3T rules
Enabled                       [VMA_UDP_3T_RULES]
 Pid:  2378 Tid:  2378 VMA DETAILS: ETH MC L2 only rules
Disabled                      [VMA_ETH_MC_L2_ONLY_RULES]
 Pid:  2378 Tid:  2378 VMA DETAILS: Force Flowtag for MC
Disabled                      [VMA_MC_FORCE_FLOWTAG]
 Pid:  2378 Tid:  2378 VMA DETAILS: Select Poll (usec)
100000                        [VMA_SELECT_POLL]
 Pid:  2378 Tid:  2378 VMA DETAILS: Select Poll OS Force
Disabled                      [VMA_SELECT_POLL_OS_FORCE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Select Poll OS Ratio
10                            [VMA_SELECT_POLL_OS_RATIO]
 Pid:  2378 Tid:  2378 VMA DETAILS: Select Skip OS
4                             [VMA_SELECT_SKIP_OS]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Drain Interval (msec)
10                            [VMA_PROGRESS_ENGINE_INTERVAL]
```

```
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Drain WCE (max)
10000                          [VMA_PROGRESS_ENGINE_WCE_MAX]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Interrupts Moderation
Enabled                         [VMA_CQ_MODERATION_ENABLE]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Moderation Count
48                              [VMA_CQ_MODERATION_COUNT]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Moderation Period (usec)
50                              [VMA_CQ_MODERATION_PERIOD_USEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ AIM Max Count
560                             [VMA_CQ_AIM_MAX_COUNT]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ AIM Max Period (usec)
250                             [VMA_CQ_AIM_MAX_PERIOD_USEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ AIM Interval (msec)
250                             [VMA_CQ_AIM_INTERVAL_MSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ AIM Interrupts Rate (per
sec) 5000
[VMA_CQ_AIM_INTERRUPTS_RATE_PER_SEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Poll Batch (max)
16                              [VMA_CQ_POLL_BATCH_MAX]
 Pid:  2378 Tid:  2378 VMA DETAILS: CQ Keeps QP Full
Enabled                         [VMA_CQ_KEEP_QP_FULL]
 Pid:  2378 Tid:  2378 VMA DETAILS: QP Compensation Level
256                             [VMA_QP_COMPENSATION_LEVEL]
 Pid:  2378 Tid:  2378 VMA DETAILS: Offloaded Sockets
Enabled                         [VMA_OFFLOADED_SOCKETS]
 Pid:  2378 Tid:  2378 VMA DETAILS: Timer Resolution (msec)
10                              [VMA_TIMER_RESOLUTION_MSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP Timer Resolution (msec)
100                             [VMA_TCP_TIMER_RESOLUTION_MSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP control thread
0 (Disabled)                    [VMA_TCP_CTL_THREAD]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP timestamp option
0                               [VMA_TCP_TIMESTAMP_OPTION]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP nodelay
0                               [VMA_TCP_NODELAY]
```

```
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP quickack
0                           [VMA_TCP_QUICKACK]
 Pid:  2378 Tid:  2378 VMA DETAILS: Exception handling mode
-1(just log debug message) [VMA_EXCEPTION_HANDLING]
 Pid:  2378 Tid:  2378 VMA DETAILS: Avoid sys-calls on tcp fd
Disabled                    [VMA_AVOID_SYS_CALLS_ON_TCP_FD]
 Pid:  2378 Tid:  2378 VMA DETAILS: Allow privileged sock opt
Enabled                     [VMA_ALLOW_PRIVILEGED_SOCK_OPT]
 Pid:  2378 Tid:  2378 VMA DETAILS: Delay after join (msec)
0                           [VMA_WAIT_AFTER_JOIN_MSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: Internal Thread Affinity
-1                          [VMA_INTERNAL_THREAD_AFFINITY]
 Pid:  2378 Tid:  2378 VMA DETAILS: Internal Thread Cpuset
[VMA_INTERNAL_THREAD_CPUSET]
 Pid:  2378 Tid:  2378 VMA DETAILS: Internal Thread Arm CQ
Disabled                    [VMA_INTERNAL_THREAD_ARM_CQ]
 Pid:  2378 Tid:  2378 VMA DETAILS: Internal Thread TCP Handling
0 (deferred)
[VMA_INTERNAL_THREAD_TCP_TIMER_HANDLING]
 Pid:  2378 Tid:  2378 VMA DETAILS: Thread mode
Multi spin lock             [VMA_THREAD_MODE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Buffer batching mode
1 (Batch and reclaim buffers) [VMA_BUFFER_BATCHING_MODE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Mem Allocate type
1 (Contig Pages)            [VMA_MEM_ALLOC_TYPE]
 Pid:  2378 Tid:  2378 VMA DETAILS: Num of UC ARPs
3                           [VMA_NEIGH_UC_ARP_QUATA]
 Pid:  2378 Tid:  2378 VMA DETAILS: UC ARP delay (msec)
10000                      [VMA_NEIGH_UC_ARP_DELAY_MSEC]
 Pid:  2378 Tid:  2378 VMA DETAILS: Num of neigh restart retries
1                           [VMA_NEIGH_NUM_ERR_RETRIES]
 Pid:  2378 Tid:  2378 VMA DETAILS: IPOIB support
Enabled                     [VMA_IPOIB]
 Pid:  2378 Tid:  2378 VMA DETAILS: SocketXtreme mode
Disabled                    [VMA_SOCKETXTREME]
```

```
 Pid:  2378 Tid:  2378 VMA DETAILS: BF (Blue Flame)
Enabled                      [VMA_BF]
 Pid:  2378 Tid:  2378 VMA DETAILS: fork() support
Enabled                      [VMA_FORK]
 Pid:  2378 Tid:  2378 VMA DETAILS: close on dup2()
Enabled                      [VMA_CLOSE_ON_DUP2]
 Pid:  2378 Tid:  2378 VMA DETAILS: MTU
0 (follow actual MTU)        [VMA_MTU]
 Pid:  2378 Tid:  2378 VMA DETAILS: MSS
0 (follow VMA_MTU)           [VMA_MSS]
 Pid:  2378 Tid:  2378 VMA DETAILS: TCP CC Algorithm
0 (LWIP)                     [VMA_TCP_CC_ALGO]
 Pid:  2378 Tid:  2378 VMA DETAILS: Polling Rx on Tx TCP
Disabled                     [VMA_RX_POLL_ON_TX_TCP]
 Pid:  2378 Tid:  2378 VMA DETAILS: Trig dummy send getsockname()
Disabled                     [VMA_TRIGGER_DUMMY_SEND_GETSOCKNAME]
VMA INFO: ---------------------------------------------------------
----------
```

Configuration Parameters Values

The following table lists the VMA configuration parameters and their possible values.

VMA Configuration Parameter	Description and Examples
VMA_TRACELEVEL	PANIC = 0 – Panic level logging. This trace level causes fatal behavior and halts the application, typically caused by memory allocation problems. PANIC level is rarely used.
	ERROR = 1 – Runtime errors in VMA. Typically, this trace level assists you to identify internal logic errors, such as errors from underlying OS or InfiniBand verb calls, and internal double mapping/unmapping of objects.
	WARN = WARNING = 2– Runtime warning that does not disrupt the application workflow.

VMA Configuration Parameter	Description and Examples
	A warning may indicate problems in the setup or in the overall setup configuration. For example, address resolution failures (due to an incorrect routing setup configuration), corrupted IP packets in the receive path, or unsupported functions requested by the user application.
	INFO = INFORMATION = 3– General information passed to the user of the application. This trace level includes configuration logging or general information to assist you with better use of the VMA library.
	DETAILS – Greater general information passed to the user of the application. This trace level includes printing of all environment variables of VMA at start up.
	DEBUG = 4 – High-level insight to the operations performed in VMA. In this logging level all socket API calls are logged, and internal high-level control channels log their activity.
	FINE = FUNC = 5 – Low-level runtime logging of activity. This logging level includes basic Tx and Rx logging in the fast path. Note that using this setting lowers application performance. We recommend that you use this level with the VMA_LOG_FILE parameter.
	FINER = FUNC_ALL = 6 – Very low-level runtime logging of activity. This logging level *drastically* lowers application performance. We recommend that you use this level with the VMA_LOG_FILE parameter.
VMA_LOG_DETAILS	Provides additional logging details on each log line. 0 = Basic log line 1 = With ThreadId 2 = With ProcessId and ThreadId 3 = With Time, ProcessId, and ThreadId (Time is the amount of milliseconds from the start of the process) Default: 0 For VMA_TRACELEVEL >= 4, this value defaults to 2.
VMA_LOG_FILE	Redirects all VMA logging to a specific user-defined file. This is very useful when raising the VMA_TRACELEVEL. The VMA replaces a single '%d' appearing in the log file name with the pid of the process loaded with VMA. This can help when running multiple instances of VMA, each with its own log file name. Example: VMA_LOG_FILE=/tmp/vma_log.txt

VMA Configuration Parameter	Description and Examples
VMA_CONFIG_FILE	Sets the full path to the VMA configuration file. Example: VMA_CONFIG_FILE=/tmp/libvma.conf Default: /etc/libvma.conf
LOG_COLORS	Uses a color scheme when logging; red for errors and warnings, and dim for very low level debugs. VMA_LOG_COLORS is automatically disabled when logging is done directly to a non-terminal device (for example, when VMA_LOG_FILE is configured). Default: 1 (Enabled)
VMA_CPU_USAGE_STATS	Calculates the VMA CPU usage during polling hardware loops. This information is available through VMA stats utility. Default: 0 (Disabled)
VMA_APPLICATION_ID	Specifies a group of rules from libvma.conf for VMA to apply. Example: VMA_APPLICATION_ID=iperf_server Default: VMA_DEFAULT_APPLICATION_ID (match only the '*' group rule)
VMA_HANDLE_SIGINTR	When enabled, the VMA handler is called when an interrupt signal is sent to the process. VMA also calls the application's handler, if it exists. Range: 0 to 1 Default: 0 (Disabled)
VMA_HANDLE_SIGSEGV	When enabled, a print backtrace is performed, if a segmentation fault occurs. Range: 0 to 1 Default: 0 (Disabled)
VMA_STATS_FD_NUM	Maximum number of sockets monitored by the VMA statistics mechanism. Range: 0 to 1024 Default: 100
VMA_STATS_FILE	Redirects socket statistics to a specific user-defined file. VMA dumps each socket's statistics into a file when closing the socket. Example: VMA_STATS_FILE=/tmp/stats
VMA_STATS_SHMEM_DIR	Sets the directory path for VMA to create the shared memory files for vma_stats. If this value is set to an empty string: “ “, no shared memory files are created. Default: /tmp/

VMA Configuration Parameter	Description and Examples
VMA_VMAD_NOTIFY_DIR	Sets the directory path for VMA to write files used by vmad. Default value is /tmp/vma Note: when used vmad must be run with --notify-dir directing the same folder.
VMA_TCP_MAX_SYN_RATE	Limits the number of TCP SYN packets that VMA handles per second for each listen socket. Example: by setting this value to 10, the maximal number of TCP connection accepted by VMA per second for each listen socket will be 10. Set this value to 0 for VMA to handle an unlimited number of TCP SYN packets per second for each listen socket. Value range is 0 to 100000. Default value is 0 (no limit)
VMA_TX_SEGS_TCP	Number of TCP LWIP segments allocation for each VMA process. Default: 1000000
VMA_TX_BUFS	Number of global Tx data buffer elements allocation. Default: 200000
VMA_TX_WRE	Number of Work Request Elements allocated in all transmit QP's. The number of QP's can change according to the number of network offloaded interfaces. Default: 3000 The size of the Tx buffers is determined by the VMA_MTU parameter value (see below). If this value is raised, the packet rate peaking can be better sustained; however, this increases memory usage. A smaller number of data buffers gives a smaller memory footprint, but may not sustain peaks in the data rate.
VMA_TX_WRE_BATCHING	Controls the number of aggregated Work Requests Elements before receiving a completion signal (CQ entry) from the hardware. Previously this number was hard coded as 64. The new update allows a better control of the jitter encountered in the Tx completion handling. Valid value range: 1-64 Default: 64
VMA_TX_MAX_INLINE	Max send inline data set for QP. Data copied into the INLINE space is at least 32 bytes of headers and the rest can be user datagram payload.

VMA Configuration Parameter	Description and Examples
	VMA_TX_MAX_INLINE=0 disables INLINEing on the TX transmit path. In older releases this parameter was called VMA_MAX_INLINE. Default: 220
VMA_TX_MC_LOOPBACK	Sets the initial value used internally by the VMA to control multicast loopback packet behavior during transmission. An application that calls setsockopt() with IP_MULTICAST_LOOP overwrites the initial value set by this parameter. Range: 0 - Disabled, 1 - Enabled Default: 1
VMA_TX_NONBLOCKED_EAGAINS	Returns value 'OK' on all send operations that are performed on a non-blocked udp socket. This is the OS default behavior. The datagram sent is silently dropped inside the VMA or the network stack. When set to Enabled (set to 1), VMA returns with error EAGAIN if it was unable to accomplish the send operation, and the datagram was dropped. In both cases, a dropped Tx statistical counter is incremented. Default: 0 (Disabled)
VMA_TX_PREFETCH_BYTES	Accelerates an offloaded send operation by optimizing the cache. Different values give an optimized send rate on different machines. We recommend that you adjust this parameter to your specific hardware. Range: 0 to MTU size Disable with a value of 0 Default: 256 bytes
VMA_RX_BUFS	The number of Rx data buffer elements allocated for the processes. These data buffers are used by all QPs on all HCAs, as determined by the VMA_QP_LOGIC. Default: 200000
VMA_RX_WRE	The number of Work Request Elements allocated in all received QPs. Default: 16000
VMA_RX_WRE_BATCHING	Number of Work Request Elements and RX buffers to batch before recycling. Batching decreases the latency mean, but might increase latency STD. Valid value range: 1-1024 Default: 64
VMA_RX_BYTES_MIN	The minimum value in bytes used per socket by the VMA when applications call to *setsockopt(SO_RCVBUF)*.

VMA Configuration Parameter	Description and Examples
	If the application tries to set a smaller value than configured in VMA_RX_BYTES_MIN, VMA forces this minimum limit value on the socket. VMA offloaded sockets receive the maximum amount of ready bytes. If the application does not drain sockets and the byte limit is reached, newly received datagrams are dropped. The application's socket usage of current, max,dropped bytes and packet counters, can be monitored using *vma_stats*. Default: 65536
VMA_RX_POLL	The number of times to unsuccessfully poll an Rx for VMA packets before going to sleep. Range: -1, 0 ... 100,000,000 Default: 100,000 This value can be reduced to lower the load on the CPU. However, the price paid for this is that the Rx latency is expected to increase. Recommended values: • 10000 – when CPU usage is not critical and Rx path latency is critical. • 0 – when CPU usage is critical and Rx path latency is not critical. • -1 – causes infinite polling. Once the VMA has gone to sleep, if it is in blocked mode, it waits for an interrupt; if it is in non-blocked mode, it returns -1. This Rx polling is performed when the application is working with direct blocked calls to read(), recv(), recvfrom(), and recvmsg(). When the Rx path has successful poll hits, the latency improves dramatically. However, this causes increased CPU utilization. For more information, see Debugging, Troubleshooting, and Monitoring.
VMA_RX_POLL_INIT	VMA maps all UDP sockets as potential Offloaded-capable. Only after ADD_MEMBERSHIP is set, the offload starts working and the CQ polling starts VMA. This parameter controls the polling count during this transition phase where the socket is a UDP unicast socket and no multicast addresses were added to it. Once the first ADD_MEMBERSHIP is called, the VMA_RX_POLL (above) takes effect. Value range is similar to the VMA_RX_POLL (above). Default: 0
VMA_RX_UDP_POLL_O	Defines the ratio between VMA CQ poll and OS FD poll.

VMA Configuration Parameter	Description and Examples
S_RATIO	This will result in a single poll of the not-offloaded sockets every VMA_RX_UDP_POLL_OS_RATIO offloaded socket (CQ) polls. No matter if the CQ poll was a hit or miss. No matter if the socket is blocking or non-blocking. When disabled, only offloaded sockets are polled. This parameter replaces the two old parameters: • VMA_RX_POLL_OS_RATIO and • VMA_RX_SKIP_OS Disable with 0 Default: 10
VMA_HW_TS_CONVERSION	Defines timestamp conversion method. The value of VMA_HW_TS_CONVERSION is determined by all devices, that is, if the hardware of one device does not support the conversion, then it will be disabled for the other devices. Currently only UDP RX flow is supported. Options = [0,1,2,3,4]: • 0 – Disabled • 1 – Raw-HW time Only convert the timestamp to seconds.nano_seconds time units (or disable if hardware does not supports). • 2 – Best possible – Raw-HW or system time. Sync to system time, then Raw hardware time. Disable if none of them are supported by hardware. • 3 – Sync to system time Convert the timestamp to seconds.nano_seconds time units. Comparable to UDP receive software timestamp. Disable if hardware does not supports. • 4 – PTP Sync Convert the timestamp to seconds.nano_seconds time units. In case it is not supported – will apply option 3 (or disable if hardware does not supports). Default value: **3** (Sync to system time)
VMA_RX_POLL_YIELD	When an application is running with multiple threads on a limited number of cores, there is a need for each thread polling inside VMA (read, readv, recv, and recvfrom) to yield the CPU to another polling thread so as not to starve them from processing incoming packets.

VMA Configuration Parameter	Description and Examples
	Default: 0 (Disabled)
VMA_RX_PREFETCH_BYTES	The size of the receive buffer to prefetch into the cache while processing ingress packets. The default is a single cache line of 64 bytes which should be at least 32 bytes to cover the IPoIB+IP+UDP headers and a small part of the user payload. Increasing this size can help improve performance for larger user payloads. Range: 32 bytes to MTU size Default: 256 bytes
VMA_RX_CQ_DRAIN_RATE_NSEC	Socket's receive path CQ drain logic rate control. When disabled (default), the socket's receive path attempts to return a ready packet from the socket's receive ready packet queue. If the ready receive packet queue is empty, the socket checks the CQ for ready completions for processing. When enabled, even if the socket's receive ready packet queue is not empty, this parameter checks the CQ for ready completions for processing. This CQ polling rate is controlled in nanosecond resolution to prevent CPU consumption due to over CQ polling. This enables improved 'real-time' monitoring of the socket ready packet queue. Recommended value is 100-5000 (nsec) Default: 0 (Disabled)
VMA_RX_POLL_ON_TX_TCP	Enables TCP RX polling during TXP TX operation for faster TCP ACK reception Default: 0 (Disabled)
VMA_GRO_STREAMS_MAX	Controls the number of TCP streams to perform GRO (generic receive offload) simultaneously. Disable GRO with a value of 0. Default: 32
VMA_TCP_3T_RULES	Uses only 3 tuple rules for TCP, instead of using 5 tuple rules. This can improve performance for a server with a listen socket which accepts many connections from the same source IP. Enable with a value of 1. Default: 0 (Disabled)
VMA_UDP_3T_RULES	This parameter is relevant in case the application uses connected UDP sockets. 3 tuple rules are used in hardware flow steering rule when the parameter is enabled, and in 5 tuple flow steering rule when it is disabled. Enabling this option can reduce hardware flow steering resources.

VMA Configuration Parameter	Description and Examples
	However, when it is disabled, the application might see benefits in latency and cycles per packet. Default: 1 (Enable)
VMA_ETH_MC_L2_ONLY_RULES	Uses only L2 rules for Ethernet Multicast. All loopback traffic will be handled by VMA instead of OS. Enable with a value of 1. Default: 0 (Disabled)
VMA_SELECT_POLL	The duration in micro-seconds (usec) in which to poll the hardware on Rx path before blocking for an interrupt (when waiting and also when calling select(), poll(), or epoll_wait()). Range: -1, 0 ... 100,000,000 Default: 100,000 When the selected path has successfully received poll hits, the latency improves dramatically. However, this comes at the expense of CPU utilization. For more information, see Debugging, Troubleshooting, and Monitoring.
VMA_SELECT_POLL_OS_RATIO	This enables polling the OS file descriptors while the user thread calls select(), poll(), or epoll_wait(), and VMA is busy in the offloaded socket polling loop. This results in a single poll of the non-offloaded sockets every VMA_SELECT_POLL_RATIO offloaded socket (CQ) polls. When disabled, only offloaded sockets are polled. (See VMA_SELECT_POLL for more information.) Disable with 0 Default: 10
VMA_SELECT_SKIP_OS	In select(), poll(), or epoll_wait()forces the VMA to check the non-offloaded sockets even though an offloaded socket has a ready packet that was found while polling. Range: 0 ... 10,000 Default: 4
VMA_CQ_POLL_BATCH_MAX	The maximum size of the array while polling the CQs in the VMA. Default: 16
VMA_PROGRESS_ENGINE_INTERVAL	Internal VMA thread safety which checks that the CQ is drained at least once every N milliseconds. This mechanism allows VMA to progress the TCP stack even when the application does not access its socket (so it does not provide a context to VMA). If the CQ was already drained by the application receive socket API calls, this thread goes back to sleep without any processing.

VMA Configuration Parameter	Description and Examples
	Disable with 0 Default: 10 milliseconds
VMA_PROGRESS_ENGINE_WCE_MAX	Each time the VMA's internal thread starts its CQ draining, it stops when it reaches this maximum value. The application is not limited by this value in the number of CQ elements that it can ProcessId from calling any of the receive path socket APIs. Default: 2048
VMA_CQ_MODERATION_ENABLE	Enable CQ interrupt moderation. Default: 1 (Enabled)
VMA_CQ_MODERATION_COUNT	Number of packets to hold before generating interrupt. Default: 48
VMA_CQ_MODERATION_PERIOD_USEC	Period in microseconds for holding the packet before generating interrupt. Default: 50
VMA_CQ_AIM_MAX_COUNT	Maximum count value to use in the adaptive interrupt moderation algorithm. Default: 560
VMA_CQ_AIM_MAX_PERIOD_USEC	Maximum period value to use in the adaptive interrupt moderation algorithm. Default: 250
VMA_CQ_AIM_INTERVAL_MSEC	Frequency of interrupt moderation adaptation. Interval in milliseconds between adaptation attempts. Use value of 0 to disable adaptive interrupt moderation. Default: 250
VMA_CQ_AIM_INTERRUPTS_RATE_PER_SEC	Desired interrupts rate per second for each ring (CQ). The count and period parameters for CQ moderation will change automatically to achieve the desired interrupt rate for the current traffic rate. Default: 5000
VMA_CQ_KEEP_QP_FULL	If disabled (default), the CQ does not try to compensate for each poll on the receive path. It uses a "debt" to remember how many WRE are missing from each QP, so that it can fill it when buffers become available.

VMA Configuration Parameter	Description and Examples
	If enabled, CQ tries to compensate QP for each polled receive completion. If there is a shortage of buffers, it reposts a recently completed buffer. This causes a packet drop, and is monitored in vma_stats. Default: 1 (Enabled)
VMA_QP_COMPENSATION_LEVEL	The number of spare receive buffer CQ holds that can be allowed for filling up QP while full receive buffers are being processed inside VMA. Default: 256 buffers
VMA_OFFLOADED_SOCKETS	Creates all sockets as offloaded/not-offloaded by default. • 1 is used for offloaded • 0 is used for not-offloaded Default: 1 (Enabled)
VMA_TIMER_RESOLUTION_MSEC	Control VMA internal thread wakeup timer resolution (in milliseconds). Default: 10 (milliseconds)
VMA_TCP_TIMER_RESOLUTION_MSEC	Controls VMA internal TCP timer resolution (fast timer) (in milliseconds). Minimum value is the internal thread wakeup timer resolution (VMA_TIMER_RESOLUTION_MSEC). Default: 100 (milliseconds)
VMA_TCP_CTL_THREAD	Does all TCP control flows in the internal thread. This feature should be disabled if using blocking poll/select (epoll is OK). • Use value of 0 to disable • Use value of 1 to wake up the thread when there is work to be done • Use value of 2 to wait for thread timer to expire Default: 0 (disabled)
VMA_TCP_TIMESTAMP_OPTION	Currently, LWIP is not supporting RTTM and PAWS mechanisms. See RFC1323 for info. • Use value of 0 to disable (enabling causing a slight performance degradation of ~50-100 nano sec per half round trip). • Use value of 1 for enable. • Use value of 2 for OS follow up. Default: 0 (disabled)

VMA Configuration Parameter	Description and Examples
VMA_TCP_NODELAY	If set, it disables the Nagle algorithm option for each TCP socket during initialization. Meaning that TCP segments are always sent as soon as possible, even if there is only a small amount of data. For more information on TCP_NODELAY flag refer to TCP manual page. Valid Values are: • 0 to disable. • 1 to enable (default)
VMA_TCP_QUICKACK	If set, it disables the delayed acknowledge ability. Meaning that TCP will respond after every packet. For more information on TCP_QUICKACK flag refer to TCP manual page. Valid Values are: • 0 to disable. • 1 to enable (default)
VMA_EXCEPTION_HANDLING	Handles missing support or error cases in Socket API or functionality by VMA. It quickly identifies VMA unsupported Socket API or features. • Use value of -1 to handle DEBUG severity • Use value of 0 to log DEBUG message and try recovering via Kernel network stack (un-offloading the socket) • Use value of 1 to log ERROR message and try recovering via Kernel network stack (un-offloading the socket) • Use value of 2 to log ERROR message and return API respectful error code • Use value of 3 to log ERROR message and abort application (throw vma_error exception). Default: -1
VMA_AVOID_SYS_CALLS_ON_TCP_FD	For TCP fd, avoid system calls for the supported options of: ioctl, fcntl, getsockopt, setsockopt. Non-supported options will go to OS. To activate, use VMA_AVOID_SYS_CALLS_ON_TCP_FD=1. Default: 0 (disabled)
VMA_THREAD_MODE	By default VMA is ready for multi-threaded applications, meaning it is thread-safe.

VMA Configuration Parameter	Description and Examples
	If the user application is single threaded, use this configuration parameter to help eliminate VMA locks and improve performance. Values: • 0 – Single-threaded application • 1 – Multi-threaded application with spin lock • 2 – Multi-threaded application with mutex lock • 3 – Multi-threaded application with more threads than cores using spin lock Default: 1 (Multi with spin lock)
VMA_BUFFER_BATCHING_MODE	Enables batching of returning Rx buffers and pulling Tx buffers per socket. • If the value is 0, then VMA will not use buffer batching • If the value is 1, then VMA will use buffer batching and will try to periodically reclaim unused buffers • If the value is 2, then VMA will use buffer batching with no reclaim Default: 1
VMA_MEM_ALLOC_TYPE	This replaces the VMA_HUGETBL parameter logic. VMA will try to allocate data buffers as configured: • 0 – "ANON" – using malloc • 1 – "CONTIG" – using contiguous pages • 2 – "HUGEPAGES" – using huge pages. OFED will also try to allocate QP & CQ memory accordingly: • 0 – "ANON" – default – use current pages ANON small ones. • "HUGE" – force huge pages • "CONTIG" – force contig pages ◦ 1 – "PREFER_CONTIG" – try contig fallback to ANON small pages. ◦ "PREFER_HUGE" – try huge fallback to ANON small pages. ▪ 2 – "ALL" – try huge fallback to contig if failed fallback to ANON small pages. To override OFED use: (MLX_QP_ALLOC_TYPE, MLX_CQ_ALLOC_TYPE). Default: 1 (Contiguous pages)

VMA Configuration Parameter	Description and Examples
VMA_FORK	Controls VMA fork support. Setting this flag on will cause VMA to call ibv_fork_init() function. ibv_fork_init() initializes *libibverbs's* data structures to handle fork() function calls correctly and avoid data corruption. If ibv_fork_init() is not called or returns a non-zero status, then *libibverbs* data structures are not fork()-safe and the effect of an application calling fork() is undefined. ibv_fork_init() works on Linux kernels 2.6.17 and later, which support the MADV_DONTFORK flag for madvise(). You should use an OFED stack version that supports fork() with huge pages (MLNX_OFED 1.5.3 to 3.2 and 4.0-2.0.0.0 and later). VMA allocates huge pages (VMA_HUGETBL) by default. For limitations of using fork() with VMA, please refer to the Release Notes. Default: 1 (Enabled)
VMA_MTU	Size of each Rx and Tx data buffer (Maximum Transfer Unit). This value sets the fragmentation size of the packets sent by the VMA library. • If VMA_MTU is 0, then for each interface VMA will follow the actual MTU • If VMA_MTU is greater than 0, then this MTU value is applicable to all interfaces regardless of their actual MTU Default: 0 (following interface actual MTU)
VMA_MSS	Defines the max TCP payload size that can be sent without IP fragmentation. Value of 0 will set VMA's TCP MSS to be aligned with VMA_MTU configuration (leaving 40 bytes of room for IP + TCP headers; "TCP MSS = VMA_MTU - 40"). Other VMA_MSS values will force VMA's TCP MSS to that specific value. Default: 0 (following VMA_MTU)
VMA_CLOSE_ON_DUP2	When this parameter is enabled, VMA handles the duplicated file descriptor (oldfd), as if it is closed (clear internal data structures) and only then forwards the call to the OS. This is, in effect, a very rudimentary *dup2* support. It supports only the case where *dup2* is used to close file descriptors. Default: 1 (Enabled)
VMA_INTERNAL_THREA	Controls which CPU core(s) the VMA internal thread is serviced on. The CPU set should be provided as either a hexadecimal value that represents

VMA Configuration Parameter	Description and Examples
D_AFFINITY	a bitmask or as a comma delimited of values (ranges are ok). Both the bitmask and comma delimited list methods are identical to what is supported by the taskset command. See the man page on taskset for additional information. The -1 value disables the Internal Thread Affinity setting by VMA. Bitmask examples: 0x00000001 – Run on processor 0 0x00000007 – Run on processors 1,2, and 3 Comma delimited examples: 0,4,8 – Run on processors 0,4, and 8 0,1,7-10 – Run on processors 0,1,7,8,9 and 10 Default: -1.
VMA_INTERNAL_THREAD_CPUSET	Selects a CPUSET for VMA internal thread (For further information, see man page of cpuset). The value is either the path to the CPUSET (for example: /dev/cpuset/my_set), or an empty string to run it on the same CPUSET the process runs on.
VMA_INTERNAL_THREAD_ARM_CQ	Wakes up the internal thread for each packet that the CQ receives. Polls and processes the packet and brings it to the socket layer. This can minimize latency for a busy application that is not available to receive the packet when it arrives. However, this might decrease performance for high pps rate applications. Default: 0 (Disabled)
VMA_INTERNAL_THREAD_TCP_TIMER_HANDLING	Selects the internal thread policy when handling TCP timers. Use value of 0 for deferred handling. The internal thread will not handle TCP timers upon timer expiration (once every 100ms) in order to let application threads handling it first. Use value of 1 for immediate handling. The internal thread will try locking and handling TCP timers upon timer expiration (once every 100ms). Application threads may be blocked till internal thread finishes handling TCP timers Default value is 0 (deferred handling)
VMA_WAIT_AFTER_JOIN_MSEC	This parameter indicates the time of delay the first packet is send after receiving the multicast JOINED event from the SM. This is helpful to overcome loss of first few packets of an outgoing stream due to SM lengthy handling of MFT configuration on the switch chips.

VMA Configuration Parameter	Description and Examples
	Default: 0 (milli-sec)
VMA_NEIGH_UC_ARP_QUATA	VMA will send UC ARP in case neigh state is NUD_STALE. If that neigh state is still NUD_STALE VMA will try VMA_NEIGH_UC_ARP_QUATA retries to send UC ARP again and then will send BC ARP. Default: 3
VMA_NEIGH_UC_ARP_DELAY_MSEC	This parameter indicates number of msec to wait between every UC ARP. Default: 10000
VMA_NEIGH_NUM_ERR_RETRIES	Indicates number of retries to restart NEIGH state machine if NEIGH receives ERROR event. Default: 1
VMA_BF	Enables/disables BlueFlame usage of the card. Default: 1 (Enabled)
VMA_SOCKETXTREME	When this parameter is enabled, VMA operates in SocketXtreme mode. SocketXtreme mode brings down latency, eliminating copy operations and increasing throughput, thus allowing applications to further utilize true kernel bypass architecture. An application should use a socket extension API named SocketXtreme. Default: 0 (Disabled)
VMA_TRIGGER_DUMMY_SEND_GETSOCKNAME	This parameter triggers dummy packet sent from getsockname() to warm up the caches. For more information see section "Dummy Send" to Improve Low Message Rate Latency. Default: 0 (Disable)
VMA_SPEC	**Note** VMA_SPEC sets all the required configuration parameters of VMA. Usually, no additional configuration is required. VMA predefined specification profile for latency:

VMA Configuration Parameter	Description and Examples
	• latency – Latency profile spec – optimized latency on all use cases. System is tuned to keep balance between Kernel and VMA. **Note** It may not reach the maximum bandwidth. Example: VMA_SPEC=latency • multi_ring_latency – Multi ring latency spec – optimized for use cases that are keen on latency where two applications communicate using send-only and receive-only TCP sockets Example: VMA_SPEC=multi_ring_latency

Beta Level Features Configuration Parameters

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

These VMA 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.

VMA Configuration Parameter	Description and Examples
VMA_RING_ALLOCATION_LOGIC_TX VMA_RING_ALLOCATIO	Ring allocation logic is used to separate the traffic into different rings. By default, all sockets use the same ring for both RX and TX over the same interface. For different interfaces, different rings are used, even when specifying the logic to be per socket or thread. Using different rings is useful when tuning for a multi-threaded application and aiming for HW resource separation.

VMA Configuration Parameter	Description and Examples
N_LOGIC_RX	**Warning** This feature might decrease performance for applications which their main processing loop is based on select() and/or poll(). The logic options are: • 0 – Ring per interface • 1 – Ring per IP address (using IP address) • 10 – Ring per socket (using socket ID as separator) • 20 – Ring per thread (using the ID of the thread in which the socket was created) • 30 – Ring per core (using CPU ID) • 31 – Ring per core - attach threads: attach each thread to a CPU core Default: 0
VMA_RING_MIGRATION_RATIO_TX VMA_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 VMA_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: 100
VMA_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. **Note** VMA_RX_BUFS might need to be adjusted in order to have enough buffers for all rings in the system. Each

VMA Configuration Parameter	Description and Examples
	ring consumes VMA_RX_WRE buffers. Use a value of 0 for an unlimited number of rings. Default: 0 (no limit)
VMA_RING_DEV_MEM_TX	VMA 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. Using VMA_RING_DEV_MEM_TX, enables the user to set the amount of the on-device-memory buffer allocated for each TX ring. 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
VMA_TCP_CC_ALGO	TCP congestion control algorithm. The default algorithm coming with LWIP is a variation of Reno/New-Reno. The new Cubic algorithm was adapted from FreeBsd implementation. Use value of 0 for LWIP algorithm. Use value of 1 for the Cubic algorithm. Use value of 2 in order to disable the congestion algorithm. Default: 0 (LWIP).

Loading VMA Dynamically

VMA 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 vma_socket;
        int fd;

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

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

        fd = vma_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 vma-redirect.h and vma-redirect.cpp files.

XLIO Parameters

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

> **Note**
>
> 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.

Commonly used parameters

The following table lists configuration parameters which can be used to tune XLIO performance for more specific use cases.

XLIO Configuration Parameter	Description and Examples
XLIO_TSO	With Segmentation Offload, or TCP Large Send, TCP can pass a buffer to be transmitted that is bigger than the maximum transmission unit (MTU) supported by the medium. Intelligent adapters implement large sends by using the prototype TCP and IP headers of the incoming send buffer to carve out segments of required size. Copying the prototype header and options, then calculating the sequence number and checksum fields creates TCP segment headers. Expected benefits: Throughput increase and CPU unload. Default value: auto (Depends on ethtool setting and adapter ability. See ethtool -k <eth0> | grep tcp-segmentation-offload) Set XLIO_TSO=1 to ensure Segmentation Offload is on for TX throughput oriented applications.
XLIO_LRO	Large receive offload (LRO) is a technique for increasing inbound throughput of high-bandwidth network connections by reducing central processing unit (CPU) overhead. It works by aggregating multiple incoming packets from a single stream into a larger buffer before they are passed higher up the networking stack, thus reducing the number of packets that must be processed. Default value: auto (Depends on ethtool setting and adapter ability. See ethtool -k <eth0> | grep large-receive-offload) Set XLIO_LRO=1 to ensure LRO is turned on for RX throughput oriented applications.
XLIO_CQ_POLL_BATCH_MAX	Max size of the array while polling the RX CQs in XLIO. Default value is 16
XLIO_GRO_STREAMS_MAX	Control the number of TCP streams to perform GRO (generic receive offload) simultaneously. Disable GRO with a value of 0. A GRO session is flushed after each RX poll cycle. See XLIO_CQ_POLL_BATCH_MAX. Default value is 32
XLIO_SELECT_POLL	The duration in micro-seconds (usec) in which to poll the hardware on Rx path before

XLIO Configuration Parameter	Description and Examples
	going to sleep (pending an interrupt blocking on OS select(), poll() or epoll_wait(). The max polling duration will be limited by the timeout the user is using when calling select(), poll() or epoll_wait(). When select(), poll() or epoll_wait() path has successful receive poll hits (see performance monitoring) the latency is improved dramatically. This comes on account of CPU utilization. Value range is -1, 0 to 100,000,000 Where value of -1 is used for infinite polling Where value of 0 is used for no polling (interrupt driven) Default value is 100000
XLIO_SELECT_POLL_OS_RATIO	This will enable polling of the OS file descriptors while user thread calls select() or poll() and the XLIO is busy in the offloaded sockets polling loop. This will result in a single poll of the not-offloaded sockets every XLIO_SELECT_POLL offloaded sockets (CQ) polls. When disabled, only offloaded sockets are polled. (See XLIO_SELECT_POLL for more info) Disable with 0 Default value is 10
XLIO_TCP_QUICKACK	If set, disable delayed acknowledge ability. This means that TCP responds after every packet. For more information on TCP_QUICKACK flag refer to TCP manual page. Valid Values are: Use value of 0 to disable. Use value of 1 for enable. Default value is Disabled.
XLIO_TCP_SEND_BUFFER_SIZE	TCP send buffer size. Default value is 1MB.
XLIO_TX_BUF_SIZE	Size of Tx data buffer elements allocation. Can not be less then MTU (Maximum Transfer Unit) and greater than 0xFF00. Default value is calculated basing on XLIO_MTU and XLIO_MSS.
XLIO_RX_POLL_ON_TX_TCP	This parameter enables/disables TCP RX polling during TCP TX operation for faster TCP ACK reception.

XLIO Configuration Parameter	Description and Examples
	Default: 0 (Disabled)
XLIO_SKIP_POLL_IN_RX	Allow TCP socket to skip CQ polling in rx socket call. 0 - Disabled 1 - Skip always 2 - Skip only if this socket was added to epoll before Default: 0 (Disabled)
XLIO_SPEC	XLIO predefined specification profiles. latency Optimized for use cases that are keen on latency. Example: XLIO_SPEC=latency
XLIO_MULTILOCK	Control locking type mechanism for some specific flows. Note that usage of Mutex might increase latency. 0 - Spin 1 - Mutex Default: 0 (Spin)
XLIO_TCP_2T_RULES	Use only 2 tuple rules for TCP connections, instead of using 5 tuple rules. This can help to overcome steering limitations for outgoing TCP connections. However, this option requires a unique local IP address per XLIO ring. In the default ring per thread configuration, this means that each thread must bind its sockets to a thread local IP address. Default: 0 (Disabled)
XLIO_RX_CQ_WAIT_CTRL	In scenarios of high scale of non blocking sockets in event driven usage such as epoll/poll/select turning on this parameter (XLIO_RX_CQ_WAIT_CTRL=1) avoids high CPU usage inside the Kernel while processing thread wakeup.

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_DEV_MEM_TX	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. Using XLIO_RING_DEV_MEM_TX, enables the user to set the amount of the on-device-memory buffer allocated for each TX ring. 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
XLIO_TCP_CC_ALGO	TCP congestion control algorithm. The default algorithm coming with LWIP is a variation of Reno/New-Reno. The new Cubic algorithm was adapted from FreeBsd implementation. Use value of 0 for LWIP algorithm. Use value of 1 for the Cubic algorithm. Use value of 2 in order to disable the congestion algorithm. Default: 0 (LWIP).

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: • db2* matches any program with a name starting with db2. • t?cp matches ttcp, etc.
<user-defined-id|*>	Specify the process ID to which the control statements appearing below this statement apply. **Note** You must also set the XLIO_APPLICATION_ID environment variable to the same value as *user-defined-id*.

Configuring Socket Transport Control

Use socket control statements to specify when libxlio will offload AF_INET/SOCK_STREAM or AF_INET/SOCK_DATAGRAM 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:

Option	Description
	• 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 *xlio*.
role	Specify one of the following roles: • 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>]|* • 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: `start-port[-end-port]` 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
```

> **Note**
>
> You must also set the XLIO parameter:
>
> ```
> XLIO_APPLICATION_ID=B1
> ```

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

Advanced Features

Packet Pacing

Packets transmitted over an offloaded socket may be rate-limited, thus, allowing granular rate control over the software defined flows. A rate-limited flow is allowed to transmit a few packets (burst) before its transmission rate is evaluated, and next packet is scheduled for transmission accordingly.

> **Note**
>
> This is a simple form of Packet Pacing supporting basic functionalities. For advanced Packing Pacing support and wide-range specification, please refer to Rivermax library.

Prerequisites

- MLNX_OFED version 4.1-x.x.x.x and above
- VMA supports packet pacing with NVIDIA® ConnectX®-5 devices.

  If you have MLNX_OFED installed, you can verify whether your NIC supports packet pacing by running:

  ```
  ibv_devinfo –v
  ```

  Check the supported pace range under the section *packet_pacing_caps* (this range is in Kbit per second).

  ```
  packet_pacing_caps:
  ```

```
qp_rate_limit_min:              1kbps
qp_rate_limit_max:              100000000kbps
```

Usage

Ø To apply Packet Pacing to a socket:

1. Run VMA with VMA_RING_ALLOCATION_LOGIC_TX=10.
2. Set the SO_MAX_PACING_RATE option for the socket:

```
uint32_t val = [rate in bytes per second];
setsockopt(fd, SOL_SOCKET, SO_MAX_PACING_RATE, &val,
sizeof(val));
```

Notes:

- VMA converts the setsockopt value from bytes per second to Kbit per second.
- It is possible that the socket may be used over multiple NICs, some of which support Packet Pacing and some do not. Hence, setting the SO_MAX_PACING_RATE socket option does not guarantee that Packet Pacing will be applied.

  In case of a failure when setting the packet pacing an error log will be printed to screen and no pacing will be done.

Precision Time Protocol (PTP)

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

When using VMA on a server running a PTP daemon, VMA 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: HCA clock available (NVIDIA® ConnectX®-4 and above)
- Set VMA_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 VMA environment parameter VMA_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*.

```
Server
$ sudo LD_PRELOAD=libvma.so VMA_HW_TS_CONVERSION=4 ./timestamping
<iface> SOF_TIMESTAMPING_RAW_HARDWARE
SOF_TIMESTAMPING_RX_HARDWARE

Client
$ LD_PRELOAD=libvma.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

> **Note**
>
> On-Device Memory is supported in ConnectX-5 adapter cards and above.

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, VMA achieves these goals by copying the WQE into the doorbell, and for small packets (<190 Bytes payload) VMA 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.

VMA 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 VMA and it is transparent to the user. The total size of the on-device-memory is limited to 256k for a single port HCA and to 128k for dual port HCA. Using VMA_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 4.1-1.0.3.0.1 and above
- NIC: NVIDIA® ConnectX®-5 and above.
- Protocol: Ethernet.

- Set VMA_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 VMA with DEBUG trace level:

```
VMA_TRACELEVEL=DEBUG LD_PRELOAD=<path to libvma.so> <command
line>
```

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

For example:

```
Pid: 30089 Tid: 30089 VMA DEBUG: ibch[0xed61d0]:245:print_val()
mlx5_0: port(s): 1 vendor: 4121 fw: 16.23.0258 max_qp_wr: 32768
on_device_memory: 131072
```

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

```
vma_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

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

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 VMA_TCP_QUICKACK parameter. TCP_QUICKACK threshold is disabled by default.

Linux Guest over Windows Hypervisor

Network virtual service client (NetVSC) exposes a virtualized view of the physical network adapter on the guest operating system. NetVSC can be configured to connect to a Virtual Function (VF) of a physical network adapter that supports an SR-IOV interface.

VMA is able to offload the traffic of the NetVSC using the SR-IOV interface, only if the SR-IOV interface is available during the application initialization.

While the SR-IOV interface is detached, VMA is able to redirect/forward ingress/egress packets to/from the NetVSC - this is done using a dedicated TAP device for each NetVSC, in addition to traffic control rules.

VMA can detect plugin and plugout events during runtime and route the traffic according to the events' type.

Prerequisites

- HCAs: NVIDIA® ConnectX®-5
- Operating systems:
    - Ubuntu 16.04, kernel 4.15.0-1015-azure
    - Ubuntu 18.04, kernel 4.15.0-1015-azure
    - RHEL 7.5, kernel 3.10.0-862.9.1.el7
- MLNX_OFED/Inbox driver: 4.5-x.x.x.x and above
- WinOF: v5.60 and above, WinOF-2: v2.10 and above
- Protocol: Ethernet
- Root/Net cap admin permissions
- VMA daemon enabled

VMA Daemon Design

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

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

For VMA daemon troubleshooting, see the Troubleshooting section.

TAP Statistics

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

```
vma_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 bond master 0x29e4260
- 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*

Using sockperf with VMA

Sockperf is VMA'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 VMA performance, please see the Tuning Guide and VMA Performance Tuning Guide.

To run a sockperf UDP test:

- To run the server, use:

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

- To run the client, use:

```
LD_PRELOAD=libvma.so sockperf <sockperf test> –i <server ip>
```

Where:

- *<server ip>* is the IP address of the server
- *<sockperf test>* is the test you want to run, for example, *pp* for the ping-pong test, *tp* for the throughput test, and so on. (Use sockperf -h to display a list of all available tests.)

To run a sockperf TCP test:

- To run the server, use:

```
LD_PRELOAD=libvma.so sockperf sr -i <server ip> --tcp
```

- To run the client, use:

```
LD_PRELOAD=libvma.so sockperf <sockperf test> –i <server ip>
--tcp
```

Example - Running sockperf Ping-pong Test

For optimal performance, please refer to Basic Performance Tuning.

1. Run sockperf server on Host A:

```
LD_PRELOAD=libvma.so sockperf sr
```

2. Run sockperf client on Host B:

```
LD_PRELOAD=libvma.so sockperf pp –i <server_ip>
```

VMA Extra API

Overview of the VMA Extra API

The information in this chapter is intended for application developers who want to use VMA's Extra API to maximize performance with VMA:

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

All socket applications are limited to the given Socket API interface functions. The VMA Extra API enables VMA 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.

VMA is designed as a dynamically-linked user-space library. As such, the VMA Extra API has been designed to allow the user to dynamically load VMA 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 VMA loaded as it did previously, or can use an application flag to decide which API to use: Socket API or VMA Extra API.

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

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

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

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

An example using the VMA Extra API can be seen in the udp_lat source code:

- Follow the '--vmarxfiltercb' flag for the packet filter logic
- Follow the '--vmazcopyread' 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 VMA Extra API

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

If the function returns with NULL, either VMA is not loaded with the application, or the VMA 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 VMA loaded.

Any non-NULL return value is a vma_api_t type structure pointer that holds pointers to the specific VMA Extra API function calls which are needed for the application to use.

It is recommended to call vma_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 libvma.conf Rules During Run-Time

Adds a libvma.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/recv ..)

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 libvma.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 libvma.conf rules.

Usually combined with the VMA_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

Packet Filtering

The packet filter logic gives the application developer the capability to inspect a received packet. You can then decide, on the fly, to keep or drop the received packet at this stage in processing.

The user's application packet filtering callback is defined by the prototype:

```
typedef vma_recv_callback_retval_t (*vma_recv_callback_t) (int fd,
size_t sz_iov, struct iovec iov[], struct vma_info_t* vma_info,
void *context);
```

This callback function should be registered with VMA by calling the VMA Extra API function register_recv_callback(). It can be unregistered by setting a NULL function pointer.

VMA calls the callback to notify of new incoming packets after the internal IP & UDP/TCP header processing, and before they are queued in the socket's receive queue.

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

Packet Filtering Callback Function Parameter	Description
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.
vma_info	Additional information on the packet and socket.

Packet Filtering Callback Function Parameter	Description
context	User-defined value provided during callback registration for each socket.

Note

The application can call all the Socket APIs from within the callback context.

Packet loss might occur depending on the application's behavior in the callback context. A very quick non-blocked callback behavior is not expected to induce packet loss.

Parameters the "iov" and "vma_info" are only valid until the callback context is returned to VMA. You should copy these structures for later use, if working with zero copy logic.

Zero Copy recvfrom()

Zero-copy revcfrom 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 recvmsg(), and MSG_VMA_ ZCOPY_FORCE
from	If not NULL, is set to the source address (same as recvfrom)	
fromlen	If not NULL, is set to the source address size (same as recvfrom).	

The flags parameter can contain the usual flags to recvmsg(), and also the MSG_VMA_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_VMA_ZCOPY is set upon exit.

If zero copy is performed (MSG_VMA_ZCOPY flag is returned), the buffer is filled with a vma_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 (*free_packets)(int s, struct vma_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 VMA 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 (from InfiniBand-to-Ethernet) 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.

Parameter	Description
rx total pack	The number of receive packets (from Ethernet to InfiniBand) associated with a TFM rule; has a log counter n.
rx total byte	The number of receive bytes (from Ethernet to InfiniBand) associated with a TFM rule; has a log counter n.

Dump fd Statistics using VMA 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 vma_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 VMA 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 VMA_SND_FLAGS_DUMMY bit in the flags parameter of send(), sendto(), sendmsg(), and sendmmsg() sockets API.

Parameters:

Parameter	Description
VMA_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, VMA_SND_FLAGS_DUMMY);
nextDummySendTime += dummySendCycleDuration; } } }
```

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

Note

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

Verifying “Dummy Send” capability in HW

“Dummy Send” feature is supported in hardware starting from ConnectX-4 NIC.

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

```
VMA_TRACELEVEL=DEBUG LD_PRELOAD=<path to libvma.so> <command
line>
```

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

For example:

```
Pid: 3832 Tid: 3832 VMA DEBUG: qpm[0x2097310]:121:configure()
Creating QP of transport type 'ETH' on ibv device 'mlx5_0' [0x201e460]
on port 1 Pid: 3832 Tid: 3832 VMA DEBUG:
qpm[0x2097310]:137:configure() HW Dummy send support for QP = 1 Pid:
3832 Tid: 3832 VMA 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 *vma_stats* tool to view the total amount of dummy-packets sent.

```
vma_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]
=========================================================
```

SocketXtreme

> **Note**
>
> Starting from VMA v8.5.x, VMA_POLL parameter is renamed to SocketXtreme.

The API introduced for this capability allows an application to remove the overhead of socket API from the receive flow data path, while keeping the well-known socket API for the control interface. Using such functionality the application has almost direct access to VMA's HW ring object and it is possible to implement a design which does not call socket APIs such as select(), poll(), epoll_wait(), recv(), recvfrom(), recvmsg(), read(), or readv().

The structures and constants are defined as shown below.

VMA Specific Events

```
typedef enum { VMA_SOCKETXTREME_PACKET = (1ULL << 32),
VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED = (1ULL << 33) }
vma_socketxtreme_events_t;
```

Parameter	Description
VMA_SOCKETXTREME_PACKET	New packet is available
VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED	New connection is auto accepted by server

VMA Buffer

```
struct vma_buff_t { struct vma_buff_t* next; void* payload;
uint16_t len; };
```

Parameter	Description
next	Next buffer (for last buffer next == NULL)
payload	Point to data
len	Data length

```
struct vma_packet_desc_t { size_t num_bufs; uint16_t total_len;
struct vma_buff_t* buff_lst; };
```

VMA Packet

Parameter	Description
total_len	Total data length
buff_lst	List of packet's buffers
len	Data length

```
struct vma_completion_t { struct vma_packet_desc_t packet;
uint64_t events; uint64_t user_data; struct sockaddr_in src; int
listen_fd; };
```

Parameter	Description
events	Set of events
user_data	User provided data • By default this field has FD of the socket • User is able to change the content using setsockopt() with level argument SOL_SOCKET and opname as SO_VMA_USER_DATA
src	Source address (in network byte order) set for VMA_SOCKETXTREME_PACKET and VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED events
listen_fd	Connected socket's parent/listen socket fd number. Valid in case VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED event is set.

Polling for VMA Completions

Syntax:

```
int (*socketxtreme_poll)(int fd, struct vma_completion_t*
completions, unsigned int ncompletions, int flags);
```

Where

- fd – file descriptor
- completions – array of completion elements

- ncompletions – number of elements in passed array
- flags – flags to control behavior (set zero)

Return values: Returns the number of ready completions during success. A negative value is returned in case of failure.

Description: This function polls the `fd` for VMA completions and returns maximum `ncompletions` - ready completions via the `completions` array. The `fd` represents a ring file descriptor. VMA completions are indicated for incoming packets and/or for other events. If VMA_SOCKETXTREME_PACKET flag is enabled in the vma_completion_t.events field the completion points to the incoming packet descriptor that can be accessed via the vma_completion_t.packet field. Packet descriptor points to the VMA buffers that contain data scattered by HW, so the data is delivered to the application with zero copy. Notice: after the application is finished with the returned packets and their buffers it must free them using free_vma_packets()/free_vma_buff() functions. If VMA_SOCKETXTREME_PACKET flag is disabled vma_completion_t.packet field is reserved. In addition to packet arrival event (indicated by VMA_SOCKETXTREME_PACKET flag) VMA also reports VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED event and standard epoll events via the vma_completion_t.events field. VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED event is reported when new connection is accepted by the server. When working with socketxtreme_poll() new connections are accepted automatically and accept (listen_socket) must not be called. VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED event is reported for the new connected/child socket (vma_completion_t.user_data refers to child socket) and EPOLLIN event is not generated for the listen socket. For events other than packet arrival and new connection acceptance vma_completion_t.events bitmask composed using standard epoll API events types. Notice: the same completion can report multiple events, for example VMA_SOCKETXTREME_PACKET flag can be enabled together with EPOLLOUT event, etc.

Getting Number of Attached Rings

Syntax:

```
int (*get_socket_rings_num)(int fd);
```

Where:

- fd – file descriptor

Return values: Returns the number of rings during success. A negative value is returned in case of failure.

Description: Returns the number of rings that are associated with socket.

Getting Ring FD

Syntax:

```
int (*get_socket_rings_fds)(int fd, int *ring_fds, int ring_fds_sz);
```

Where:

- fd – file descriptor
- ring_fds – int array of ring fds
- ring_fds_sz – size of the array

Return values: Returns the number populated array entries during success. A negative value is returned in case of failure.

Description: Returns FDs of the rings that are associated with the socket.

Free VMA Packets

Syntax:

```
int (*socketxtreme_free_vma_packets)(struct vma_packet_desc_t
*packets, int num);
```

Where:

- packets – packets to be freed

- num – number of packets in passed array

Return values: Returns zero value during success. A negative value is returned in case failure.

Description: Frees packets received by socketxtreme_poll().

For each packet in the `packets` array this function updates the receive queue size and the advertised TCP window size, if needed, for the socket that received the packet and frees VMA buffer list that is associated with the packet. Notice: for each buffer in the buffer list VMA decreases buffer's ref count and only buffers with ref count zero are deallocated. An application can call socketxtreme_ref_vma_buf() to increase the buffer reference count in order to hold the buffer even after socketxtreme_free_vma_packets() has been called. Also, the application is responsible to free buffers that could not be deallocated during socketxtreme_free_vma_packets() due to non-zero reference count. This is done by calling the socketxtreme_free_vma_buff() function.

Decrement VMA Buffer Reference Counter

Syntax:

```
int (*socketxtreme_free_vma_buff)(struct vma_buff_t *buff);
```

Return values: Returns the buffer's reference count after the change (zero value means that the buffer has been deallocated). A negative value is returned in case of failure.

Description: Decrement the reference counter of a buffer received by socketxtreme_poll(). This function decrements the buff reference count. When buff's reference count reaches zero, it is deallocated.

Increment VMA Buffer Reference Counter

Syntax:

```
int (*socketxtreme_ref_vma_buff)(struct vma_buff_t *buff);
```

Where:

- buff – buffer to be managed

Return values: Returns buffer's reference count after the change. A negative value is returned in case of failure.

Description: Increment the reference counter of a buffer received by socketxtreme_poll(). This function increments the reference count of the buffer. This function should be used in order to hold the buffer even after a call to socketxtreme_free_vma_packets(). When the buffer is no longer required it should be freed via socketxtreme_free_vma_buff ().

Usage Example

Sockperf benchmark supports socketxtreme mode. Its source code can be used as a reference of socketxtreme API usage.

The following sample implements server side logic based on the API described above.

In this example, the application just waits for connection requests and accepts new connections.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <mellanox/vma_extra.h>
```

```
int main(int argc, char**argv)
{
      int rc = 0;
      int fd = -1;
      struct sockaddr_in addr;
      static struct vma_api_t *_vma_api = NULL;
      static int _vma_ring_fd = -1;
      char *strdev = (argc > 1 ? argv[1] : NULL);
      char *straddr = (argc > 2 ? argv[2] : NULL);
      char *strport = (argc > 3 ? argv[3] : NULL);

      if (!strdev || !straddr || !strport) {
            printf("Wrong options\n");
            exit(1);
      }
      printf("Dev: %s\nAddress: %s\nPort:%s\n", strdev, straddr, strport);

      _/* Get VMA extra API reference */
      _vma_api = vma_get_api();
      if (_vma_api == NULL) {
            printf("VMA Extra API not found\n");
            exit(1);
      }

      fd = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);

      rc = setsockopt(fd, SOL_SOCKET, SO_BINDTODEVICE,
                      (void *)strdev, strlen(strdev));
      if (rc < 0) {
          printf("setsockopt() failed %d : %s\n", errno, strerror(errno));
          exit(1);
      }

      bzero(&addr, sizeof(addr));
      addr.sin_family = AF_INET;
      addr.sin_addr.s_addr = inet_addr(straddr);
```

```
     addr.sin_port = htons(atoi(strport));

     rc = bind(fd, (struct sockaddr *)&addr, sizeof(addr));
     if (rc < 0) {
          fprintf(stderr, "bind() failed %d : %s\n", errno,
strerror(errno));
          exit(1);
     }

     _/* RX ring is available after bind() */
     _vma_api->get_socket_rings_fds(fd, &_vma_ring_fd, 1);
     if (_vma_ring_fd == -1){
          printf("Failed to return the ring fd\n");
          exit(1);
     }

     listen(fd, 5);
     printf("Waiting on: fd=%d\n", fd);

     while (0 == rc) {
          struct vma_completion_t vma_comps;
          /* Polling any RX events / data */
          rc = _vma_api->socketxtreme_poll(_vma_ring_fd,
&vma_comps, 1, 0);
          if (rc > 0) {
               printf("socketxtreme_poll: rc=%d event=0x%lx user_data=%ld\n",
                    rc, vma_comps.events, vma_comps.user_data);
               if (vma_comps.events &
VMA_SOCKETXTREME_NEW_CONNECTION_ACCEPTED) {
                    printf("Accepted connection: fd=%d\n",
(int)vma_comps.user_data);
                    rc = 0;
               }
          }
     }
```

```
    close(fd);
    fprintf(stderr, "socket closed\n");

    return 0;
```

Installation

For instructions on how to install SocketXtreme, please refer to section Installing VMA with SocketXtreme .

Limitations

No support for:

- Multi-thread
- ConnectX-3/ConnectX-3 Pro HCAs
- MLNX_OFED version lower than v3.4

User should keep in mind the differences in flow between the standard socket API and that based on the polling completions model.

- SocketXtreme mode is used with non-blocking connect() call only
- Do not use accept() because socketxtreme_poll() provides special event as VMA_socketxtreme_NEW_CONNECTION_ACCEPTED to track connection request
- Mixed receive methods (recv/recvfrom/recmsg/epoll_wait with socketXtreme) can cause the user to receive out-of-order packets. UDP is an unreliable protocol, hence working with mixed receive methods are allowed yet not recommended. Whereas TCP is a reliable protocol, hence mixed receive methods are not allowed. socketxtreme_poll() is able to notify about any received data using the event VMA_socketxtreme_PACKET.

Monitoring, Debugging, and Troubleshooting

Monitoring – the vma_stats Utility

Networking applications open various types of sockets.

The VMA 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 VMA during select(), poll(), or epoll_wait() calls. VMA calculates this counter only if VMA_CPU_USAGE_STATS parameter is enabled, otherwise this counter is not in use and displays the default value as zero.
- VMA internal CQ performance counters
- VMA internal RING performance counters
- VMA internal Buffer Pool performance counters

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

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

Usage:

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

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

Parameter Name	Description	Values
-p, --pid	<pid>	Shows VMA statistics for a process with pid: <pid>.
-k, --directory	<directory>	Sets shared memory directory path to <directory>
-n, --name	<application>	Shows VMA statistics for application: <application>
-f, --find_pid		Finds PID and shows statistics for the VMA 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: 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: 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.

Parameter Name	Description	Values
-z, --zero		Zero counters.
-l, --log_level	<level>	Sets the VMA log level to <level> (1 <= level <= 7).
-D, --details_level	<level>	Sets the VMA 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 vma_stats utility.

Example 1

Description

The following example demonstrates basic use of the vma_stats utility.

Command Line

```
#vma_stats –p <pid>
```

Note

If there is only a single process running over VMA, it is not necessary to use the –p option, since vma_stats will automatically recognize the process.

Output

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

```
vmastat: 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
-------------------------------------------------------------------
---------
```

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 VMA_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

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

Command Line

```
#vma_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
```

```
 19  Rx:       15186        29           0          0      0.0          0          0
0
     Tx:       15186        29           0          0                   0          0
0
 23  Rx:           0         0           0          0      0.0      15185         22
0
     Tx:           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 VMA_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 VMA and the application. For example when the CPU usage is 100%, 70% is used by VMA 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 vma_stats output.

Command Line

```
#vma_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]
==================================================
        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
=========================================================
```

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 VMA_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 VMA_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 VMA_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)

Example 4

Description

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

Command Line

```
#vma_stats –p <pid> -v 4
```

Output

```
VMA 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 vma_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 VMA
- 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 VMA_STATS_FILE parameter).

Output

```
VMA: [fd=10] Tx Offload: 455 / 233020 / 0
[kilobytes/packets/errors]
VMA: [fd=10] Tx OS info: 0 / 0 / 0 [kilobytes/packets/errors]
VMA: [fd=10] Rx Offload: 455 / 233020 / 0
[kilobytes/packets/errors]
VMA: [fd=10] Rx OS info: 0 / 0 / 0 [kilobytes/packets/errors]
VMA: [fd=10] Rx byte: max 200 / dropped 0 (0.00%) / limit 2000000
VMA: [fd=10] Rx pkt : max 1 / dropped 0 (0.00%)
VMA: [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 VMA_RX_POLL and VMA_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 VMA_RX_BYTES_MIN). A single socket with user fd=14 was created.

Example 7

Description

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

Command Line

```
#vma_stats –-fd_dump 17 info
```

Output

```
VMA INFO: ============ DUMPING FD 17 STATISTICS ============
VMA INFO: ==================== SOCKET FD ====================
VMA INFO: Fd number : 17
VMA INFO: Bind info : 22.0.0.4:58795
VMA INFO: Connection info : 22.0.0.3:6666
VMA INFO: Protocol : PROTO_TCP
VMA INFO: Is closed : false
VMA INFO: Is blocking : true
VMA INFO: Rx reuse buffer pending : false
VMA INFO: Rx reuse buffer postponed : false
VMA INFO: Is offloaded : true
VMA INFO: Tx Offload : 12374 / 905105 / 0 / 0
[kilobytes/packets/drops/errors]
```

```
VMA INFO: Rx Offload : 12374 / 905104 / 0 / 0
[kilobytes/packets/eagains/errors]
VMA INFO: Rx byte : max 14 / dropped 0 (0.00%) / limit 0
VMA INFO: Rx pkt : max 1 / dropped 0 (0.00%)
VMA INFO: Rx poll : 0 / 905109 (100.00%) [miss/hit]
VMA INFO: Socket state : TCP_SOCK_CONNECTED_RDWR
VMA INFO: Connection state : TCP_CONN_CONNECTED
VMA INFO: Receive buffer : m_rcvbuff_current 0, m_rcvbuff_max
87380, m_rcvbuff_non_tcp_recved 0
VMA INFO: Rx lists size : m_rx_pkt_ready_list 0,
m_rx_ctl_packets_list 0, m_rx_ctl_reuse_list 0
VMA INFO: PCB state : ESTABLISHED
VMA INFO: PCB flags : 0x140
VMA INFO: Segment size : mss 1460, advtsd_mss 1460
VMA INFO: Window scaling : ENABLED, rcv_scale 7, snd_scale 7
VMA INFO: Receive window : rcv_wnd 87380 (682), rcv_ann_wnd 87240
(681), rcv_wnd_max 87380 (682), rcv_wnd_max_desired 87380 (682)
VMA INFO: Send window : snd_wnd 87168 (681), snd_wnd_max 8388480
(65535)
VMA INFO: Congestion : cwnd 1662014
VMA INFO: Receiver data : rcv_nxt 12678090, rcv_ann_right_edge
12765330
VMA INFO: Sender data : snd_nxt 12678080, snd_wl1 12678076, snd_wl2
12678066
VMA INFO: Send buffer : snd_buf 255986, max_snd_buff 256000
VMA INFO: Retransmission : rtime 0, rto 3, nrtx 0
VMA INFO: RTT variables : rttest 38, rtseq 12678066
VMA INFO: First unsent : NULL
VMA INFO: First unacked : seqno 12678066, len 14, seqno + len
12678080
VMA INFO: Last unacked : seqno 12678066, len 14, seqno + len 12678080
VMA INFO: Acknowledge : lastack 12678066
VMA INFO: ================================================
VMA 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 VMA
- 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 unacked queue (seqno 12678066)
- The last acknowledge sequence number is 12678066

Additional information about the values can be found in the VMA's wiki page.

Debugging

VMA Logs

Use the VMA logs in order to trace VMA operations. VMA logs can be controlled by the VMA_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 VMA_TRACELEVEL variable value up to 6 (as described in VMA Configuration Parameters) to see more information about each thread's operation. Use the VMA_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 VMA_LOG_FILE=/tmp/my_file.log to save the daily events. It is recommended to check these logs for any VMA warnings and errors. Refer to the Troubleshooting section to help resolve the different issues in the log.

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

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

Use the VMA_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 VMA (Rx and Tx).

tcpdump

For tcpdump to capture offloaded traffic (on ConnectX-4 and above), 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 VMA processes. Upon VMA 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 VMA, and describes solutions for these problems.

1. High Log Level

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

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

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

   Solution: Set VMA_TRACELEVEL to its default value 3.

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

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

Solution: Check that libvma is properly installed, and that libvma.so is located in */usr/lib* (or in */usr/lib64*, for 64-bit machines).

3. On attempting to install *vma rpm*, the following error is reported:

```
#rpm –ivh libvma-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:

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

```
VMA WARNING:
*************************************************************
VMA WARNING: * NO IMMEDIATE ACTION NEEDED!
VMA WARNING: * Not enough hugepage resources for VMA memory
allocation.
VMA WARNING: * VMA will continue working with regular memory
allocation.
VMA INFO:    * Optional:
VMA INFO:    *   1. Switch to a different memory allocation
type
VMA INFO:    *      (VMA_MEM_ALLOC_TYPE!= 2)
VMA INFO:    *   2. Restart process after increasing the
number of
VMA INFO:    *      hugepages resources in the system:
VMA INFO:    *      "echo 1000000000 > /proc/sys/kernel/shmmax"
VMA INFO:    *      "echo 800 > /proc/sys/vm/nr_hugepages"
VMA WARNING: * Please refer to the memory allocation section
in the VMA's
VMA WARNING: * User Manual for more information
VMA WARNING:
*************************************************************
```

This warning message means that you are using VMA with huge page memory allocation enabled (VMA_MEM_ALLOC_TYPE=2), but not enough huge page resources are available in the system. VMA will use contiguous pages instead.

Note that VMA_MEM_ALLOC_TYPE= 1 is not supported while working with Microsoft hypervisor. In this case – please use VMA_MEM_ALLOC_TYPE= 0 (malloc).

If you want VMA to take full advantage of the performance benefits of huge pages, restart the application after adding more huge page resources to your system similar to the details in the warning message above, 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 VMA, you will probably need to increase the values used in the above example.

> **Warning**
>
> Check that your host machine has enough free memory after allocating the huge page resources for VMA. Low system memory resources may cause your system to hang.

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

Use the following script to release VMA 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) VMA 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. VMA process cannot establish connection with daemon (vmad) in Microsoft hypervisor environment.

   When working with Microsoft Hypervisor, VMA 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 VMA startup:

```
VMA WARNING:
*************************************************************
VMA WARNING: * Can not establish connection with the daemon
(vmad).    *
```

```
VMA WARNING: * UDP/TCP connections are likely to be limited.
*
VMA WARNING:
************************************************************
```

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

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

To fix this, run “vmad” as root.

8. VMA cannot offload traffic when RoCE LAG is enabled.

   RoCE LAG is a feature meant for mimicking Ethernet bonding for IB devices and is available for dual port cards only. When 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.

   The following warning appears upon VMA startup:

```
VMA WARNING:
************************************************
VMA WARNING: * Interface bond0 will not be offloaded.
VMA WARNING: * VMA cannot offload the device while RoCE LAG is
enabled.
VMA WARNING: * Please refer to VMA Release Notes for more info
VMA WARNING:
************************************************
VMA WARNING:
************************************************
VMA WARNING: * Bond bond0 will not be offloaded due to
problem with its slaves.
VMA WARNING: * Check warning messages for more information.
```

```
VMA WARNING:
**************************************************
```

RoCE LAG should be disabled in order for VMA to be able to offload traffic.

ConnectX-4 and above HCAs with MLNX_OFED print the following warning with instructions on how to disable RoCE LAG:

```
VMA WARNING:
**************************************************
VMA WARNING: * Interface bond1 will not be offloaded.
VMA WARNING: * VMA cannot offload the device while RoCE LAG is
enabled.
VMA WARNING: * In order to disable RoCE LAG please use:
VMA WARNING: * echo 0 >
/sys/class/net/ens4f1/device/roce_lag_enable
VMA WARNING:
**************************************************
VMA WARNING:
**************************************************
VMA WARNING: * Bond bond1 will not be offloaded due to
problem with its slaves.
VMA WARNING: * Check warning messages for more information.
VMA WARNING:
**************************************************
```

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

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

```
VMA:  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

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

Sockperf can be used natively, or with VMA acceleration.

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.

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

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

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

Sockperf is installed on the VMA server at */usr/bin/sockperf*. For examples of running sockperf, see:

- Latency with Ping-pong Test

- Bandwidth and Packet Rate With Throughput Test

Note

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, sockperf presents additional advanced statistics and analysis information:

- In addition to the average latency and standard deviation, sockperf 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.

- Sockperf 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, sockperf 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 want 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
```

where *eth0* is the Ethernet interface.

You can also set the interface on runtime in sockperf:

- 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)

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).

Note

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 VMA

To run TCP ping-pong using VMA:

1. Run the server by using:

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

2. Run the client by using:

```
# VMA_SPEC=latency LD_PRELOAD=libvma.so sockperf pp -i
<server-ip> --tcp –m 64
```

Where *VMA_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>
```

2. Run the server by using:

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

3. Run the client by using:

```
# sockperf tp -i <server-100g-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)
```

Note

For more sockperf throughput options run:

```
# sockperf tp –h
```

UDP MC Throughput using VMA

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=libvma.so sockperf sr -i <server-ip>
```

2. Run the client by using:

```
# LD_PRELOAD=libvma.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 + VMA
Message Rate	851796 [msg/sec]	4228326 [msg/sec]
Bandwidth	1195.759 MBps (9566.068 Mbps)	5935.760 MBps (47486.083 Mbps)
VMA 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>.
-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).

Short Command	Full Command	Description
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 address of interface on which to receive multicast packets (can be different from the route table).
N/A	--mc-tx-if	IP address of interface on which to transmit multicast packets (can be different from the route table).
N/A	--mc-loopback-enable	Enable MC loopback (default disabled).
N/A	--mc-ttl	Limit the lifetime of the message (default 2).
N/A	--mc-source-filter	Set the address <ip, hostname> of the mulitcast 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_loopin	Set sockperf to loop over recvfrom() until EAGAIN or <N> good received packets, -1 for infinite, must be used with --nonblocked

Short Command	Full Command	Description
	g_num	(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	--vmazcopyread	If possible use VMA's zero copy reads API (see the VMA readme).
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-vma	Load VMA dynamically even when LD_PRELOAD was not used.
N/A	--rate-limit	Use rate limit (packet-pacing). When used with VMA, it must be run with VMA_RING_ALLOCATION_LOGIC_TX mode.
N/A	--set-sock-accl	Set socket acceleration before running VMA (available for some NVIDIA® systems).
-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	--vmarxfiltercb	If possible use VMA's receive path packet filter callback API (See the VMA 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.

Short Command	Full Command	Description
-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.

SocketXtreme

sockperf v3.2 and above supports VMA socketXtreme polling mode.

In order to support socketXtreme, sockperf should be configured using --enable-vma-api parameter compiled with the compatible vma_extra.h file during compilation.

New iomux type should appear -x / --socketxtreme:

Short Command	Full Command	Description
-F	--iomux-type	Type of multiple file descriptors handle [s|select|p|poll|e|epoll|r|recvfrom|x|socketxtreme](default epoll).

Note

SocketXtreme should be also enabled for VMA. For further information, please refer to Installing VMA with SocketXtreme.

In order to use socketXtreme, VMA should also be compiled using --enable-socketxtreme parameter.

socketXtreme requires forcing the Client side to bind to a specific IP address. Hence, while running UDP client with socketXtreme, running `--client_ip` is mandatory:

```
--client_ip            -Force the client side to bind to a
specific ip address (default = 0).
```

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 `--mps` and/or `--reply-every` switch, try other values or try the default values

2. If the following error is received, it means that Sockperf is trying to compile against VMA with no socketXtreme support:

```
In file included from src/Client.cpp:32:0:
src/IoHandlers.h: In member function 'int IoSocketxtreme::waitArrival()':
src/IoHandlers.h:421:71: error: 'VMA_SOCKETXTREME_PACKET' was not
declared in this scope
      if (m_rings_vma_comps_map_itr->second-
>vma_comp_list[i].events & VMA_SOCKETXTREME_PACKET){

^
src/IoHandlers.h:422:18: error: 'struct vma_api_t' has no member
named 'socketxtreme_free_vma_packets'
       g_vma_api-
>socketxtreme_free_vma_packets(&m_rings_vma_comps_map_itr-
>second->vma_comp_list[i].packet, 1);
```

There are two ways to solve this:

- Configure sockperf with --disable-vma-api parameter;

  or

- Use VMA 8.5.1 or above

Multicast Interface Definitions

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 for the receive path multicast join request, it defines the interface through which it wants the VMA 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 for the receive path multicast join request, it defines the interface through which it wants the VMA 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 on the transmit path, it defines the interface through which the VMA 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 VMA 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
```

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

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

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.

Glossary

Acronym	Definition
API	Application Programmer's Interface
CQ	Completion Queue
FD	File Descriptor
GEth	Gigabit Ethernet Hardware Interface
HCA	Host Channel Adapter
HIS	Host Identification Service
IB	InfiniBand
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPoIB	IP over IB
IPR	IP Router
NIC	Network Interface Card
OFED	OpenFabrics Enterprise Distribution
OS	Operating System
pps	Packets Per Second
QP	Queue Pair
RMDS	Reuters Market Data System
RTT	Route Trip Time
SM	Subnet Manager
UDP	User Datagram Protocol
usec	microseconds
UMCAST	User Mode Multicast
VMA	NVIDIA® Messaging Accelerator
VMS	VMA Messaging Service

Acronym	Definition
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	*/opt/ufm/conf/gv.cfg*	
Console session (code)	-> flashClear <CR>	Complete sample line or block. Comprises both input and output. The code can also be shaded.
Linux shell prompt	#	The "#"character stands for the Linux shell prompt.
CLI Guest Mode	Switch >	CLI Guest Mode.
CLI admin mode	Switch #	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,	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).

Term, Construct, Text Block	Example	Notes
item name		
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.

User Manual Revision History

Revision	Date	Description
9.8.60	February 06, 2025	• Added Installing VMA from a Dedicated DOCA Profile section • Moved VMA Installation Options under Installing VMA section
	May 06, 2024	Updated examples across the document to reflect the new 9.8.60 VMA version
9.8.51	February 11, 2024	Updated examples across the document to reflect the new 9.8.51 VMA version
9.8.40	November 2024	Updated examples across the document to reflect the new 9.8.40 VMA version
9.8.31	August 10, 2023	Updated examples across the document to reflect the new 9.8.31 VMA version
9.8.20	May 19, 2023	Updated examples across the document to reflect the new 9.8.20 VMA version
9.8.1	January 31, 2021	Updated examples across the document to reflect the new 9.8.1 VMA version
9.7.2	November 30, 2022	Updated examples across the document to reflect the new 9.7.2 VMA version
9.7.0	October 31, 2022	Updated examples across the document to reflect the new 9.7.0 VMA version
9.6.4	July 31, 2022	Updated examples across the document to reflect the new 9.6.4 VMA version
9.5.2	May 3, 2022	Updated examples across the document to reflect the new 9.5.2 VMA version
Rev 9.4.0	November 30, 2021	• Updated examples to reflect the 9.4.0 VMA version • Updated the examples in Configuring VMA

Revision	Date	Description
Rev 9.3.1	June 30, 2021	• Added section Running VMA using non-root Permission • Updated examples to reflect the 9.3.1 VMA version
Rev 8.9.4	October 02, 2019	Removed the following sections: • Multi-Packet Receive Queue • Installing VMA with SocketXtreme
Rev 8.8.3	April 30, 2019	Updated the example in Configuring VMA
		Added VMA_SOCKETXTREME entry to VMA Configuration v9.4.0 table
		Added issue #9 under Troubleshooting section

Release Notes Revision History

Release Notes Change History

Release	Description
9.8.51	• Updated MLNX_OFED and Firmware versions. See System Requirements and Interoperability • Added full support of TCP_KEEPALIVE option • See Bug Fixes History section.
9.8.40	• Removed DPCP dependency from VMA. • See Bug Fixes History section.
9.8.31	• See Bug Fixes History section.

9.5.2	• Changed the default visibility of socIPoIB is temporarily unavailable when ket API symbols by hiding internal symbols and enabling export-only functions. This change in the default library configuration helps to:
	  ◦ Substantially improve the load time of the library
	  ◦ Produce better code quality by the optimizer
	  ◦ Reduces chances of symbol collision
	• Product source code is migrated to C++11 standard requirements.
	• See Bug Fixes section.

9.4.0	• Updated Certified Applications. • Updated Known Issues. • See Bug Fixes section.

Release	Description
9.2.2	• Added RoCE LAG support to VMA over MLNX_OFED RDMA-Core. • Bug Fixes.
9.1.1	• Added man pages for libvma. • Added support for UDP 5 tuple hardware flow steering rules. • Added support for a new environment variable VMA_UDP_3T_RULES. • Removed support for ConnectX-3 and ConnectX-3 Pro NICs. • See Bug Fixes section.
9.0.2	• Added the option for VMA daemon to set spoofed SYN retry interval. • See Bug Fixes section.
8.9.4	See Bug Fixes section.
8.8.3	• Added SocketXtreme API support to the same VMA binary as the traditional Socket API. • Added the ability to specify ring allocation logic for any socket type. • Improved the VMA service installation under different Linux service managers.
8.7.5	• Added support for TCP Rx timestamping. • Added support for an additional ring allocation logic: Ring logic per IP **Usage**: `VMA_RING_ALLOCATION_LOGIC_RX/TX = 1` • Added support for IP_TTL socket option. • Added support for re-establishing lost connection with VMA daemon. • Added support for sendfile() and sendfile64() functions, where in_fd is a file descriptor open for reading, and out_fd is an offloadable socket. • Added support for IPoIB in upstream/inbox drivers for ConnectX-4 and above adapter cards.

Bug Fixes History

The following table lists the issues that have been resolved in previous VMA versions.

Internal Ref. Number	**Details**
3591039	**Description:** Fixed RX buffer leak in case of GRO and out of order packets. Fixed TCP stream corruption in case of out of order packets
	Keywords: TCP, corruption, out of order. GRO
	Discovered in Version: 8.4.101
	Fixed in Version: 9.8.40
3604175	**Description:** Fixed VMA hanging infinitely while closing ring with empty RQ
	Keywords: ring, termination, stuck, hang up
	Discovered in Version: 9.8.30
	Fixed in Version: 9.8.40
3525812	**Description:** Fixed SocketXtreme RX buffer leak.
	Keywords: SocketXtreme; leak
	Discovered in Version: 9.8.20
	Fixed in Version: 9.8.31
3373882	**Description:** Fixed a compilation error with gcc 13.0.1.
	Keywords: gcc compilation error
	Discovered in Version: 9.8.1
	Fixed in Version: 9.8.20
3173318	**Description**: Fixed the issue where VMA Debian package installation for Docker container failed.
	Keywords: Debian; Docker; Installation
	Discovered in Version: 9.7.0
	Fixed in Version: 9.7.2
3173318	**Description**: Fixed the issue where using send* functions with null elements in iov Tx vector causes an API error.
	Keywords: iov Tx vector
	Discovered in Version: 9.6.4
	Fixed in Version: 9.7.0
3092554	**Description:**

Internal Ref. Number	**Details**
	When VMA_HANDLE_SIGINTR=0, the following issues are no longer encountered: 1. When SIGNIT 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.
	Keywords: SIGINT; EINTR; signal; sigaction
	Discovered in Version: 9.5.2
	Fixed in Version: 9.6.4
3092555	**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 error is received upon second attempt instead.
	Keywords: connect(); blocking socket; segmentation fault
	Discovered in Version: 9.5.2
	Fixed in Version: 9.6.4
3045735	**Description:** Fixed the issue where there was no traffic as long as SR-IOV mode was disabled.
	Keywords: SR-IOV; traffic
	Discovered in Version: 9.3.1
	Fixed in Version: 9.6.4
2740920	**Description:** Added support for fortified glibc functions as __read_chk, __recv_chk, __recvfrom_chk, __poll_chk, __ppoll_chk.
	Keywords: socket API
	Discovered in Version: 9.3.0
	Fixed in Version: 9.4.0
1714768	**Description:** Fixed memory leak in vma_free_packets() implementation.
	Keywords: Extra API
	Discovered in Version: 8.8.2
	Fixed in Version: 9.4.0
2366027	**Description:** Fixed big-endian support for TIMESTAMP option.

Internal Ref. Number	**Details**
	Keywords: TCP
	Discovered in Version: 9.0.2
	Fixed in Version: 9.3.1
2280628	**Description**: Added TIMESTAMP option into keepalives and zero window probes TCP packets.
	Keywords: TCP
	Discovered in Version: 9.0.2
	Fixed in Version: 9.3.1
2246994	**Description**: Set proper FIN/RST flags for splitted TCP segments.
	Keywords: TCP
	Discovered in Version: 9.1.1
	Fixed in Version: 9.3.1
2130901	**Description**: Fixed forever loop condition during finalization after setting VMA_PROGRESS_ENGINE_WCE_MAX=0.
	Keywords: Hangup
	Discovered in Version: 9.0.2
	Fixed in Version: 9.3.1
1775713	**Description**: Fixed a synchronization issue in attach/detach flow when VMA is configured to use 3tuple (software rule).
	Keywords: Cleanup
	Discovered in Release: 8.4.10
	Fixed in Version: 9.2.2
2233349	**Description**: Fixed wrong detection of huge pages with different sizes.
	Keywords: Huge Page
	Discovered in Version: 9.1.1
	Fixed in Version: 9.2.2
2355289	**Description**: Fixed wrong detection of Blue Flame usage capability.
	Keywords: Blue Flame

Internal Ref. Number	**Details**
	Discovered in Version: 9.1.1
	Fixed in Version: 9.2.2
2132032	**Description**: Fixed an issue where all traffic was received in one top socket when several sockets were bound to the same IP:PORT pair and used 5 different tuple rules.
	Keywords: UDP steering
	Discovered in Version: 9.0.2
	Fixed in Version: 9.1.1
2009931	**Description**: Added fcntl64() support.
	Keywords: Socket API
	Discovered in Version: 8.9.5
	Fixed in Version: 9.1.1
2074332	**Description**: Fixed the issue where vma_stats utility reported wrong statistics.
	Keywords: vma_stats
	Discovered in Version: 9.0.1
	Fixed in Version: 9.1.1
1973965	**Description**: Replaced dropped packets statistics data with EAGAIN.
	Keywords: vma_stats
	Discovered in Version: 8.9.5
	Fixed in Version: 9.1.1
1900224	**Description:** Fixed the issue where negative values were displayed by `vma_stats` for Send queue size during long duration sessions.
	Keywords: vma_stats
	Discovered in Version: 8.9.2
	Fixed in Version: 9.1.1
1565428	**Description:** Fixed the issue where rdma_lib_reset function was not supported on the Upstream driver, resulting in fork() function being unsupported.

Internal Ref. Number	**Details**
	Keywords: rdma_lib_reset, fork(), Upstream driver
	Discovered in Version: 8.7.5
	Fixed in Version: 9.1.1
2069198	**Description**: Disabled Blue Flame (BF) operation usage for Azure.
	Keywords: Azure
	Discovered in Version: 8.9.3
	Fixed in Version: 9.0.2
1794728	**Description**: Fixed an issue related calling `unregister_timer_event()` twice.
	Keywords: Hangup
	Discovered in Version: 8.8.3
	Fixed in Version: 9.0.2
1264894	**Description:** Fixed cleanup issues when not all internal objects related sockets are destroyed during VMA finalization.
	Keywords: Cleanup
	Discovered in Version: 8.5.2
	Fixed in Version: 9.0.2