Connect. Accelerate. Outperform.™

Mellanox Messaging Accelerator (VMA) Library for Linux

User Manual

Rev 6.9.1

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT (“PRODUCT(S)”) AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES “AS-IS” WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

List of Tables

Document Revision History

Table 1: Document Revision History

Version	Description

Rev 6.9.1	• Updated the following sections:
	  • VMA Configuration Parameters
	  • Configuration Parameter Values

Rev 6.8.3	• Updated the following sections:
	  • Overview of the VMA Extra API
	  • Zero Copy recvfrom()
	  • Freeing Zero Copied Packet Buffers

Rev 6.7	• No changes

Rev 6.6.4	• Updated the following sections:
	  • Configuring Socket Transport Control
	  • VMA Configuration Parameters
	  • Configuration Parameter Values
	  • Monitoring – the vma_stats Utility
	  • Example 3

Rev 6.5.9	• Added the following sections:
	  • Adding libvma.conf Rules During Run-Time
	  • Creating Sockets as Off-loaded or Not-Off-loaded

Rev 6.4.11	• Added the following sections:
	  • Using sockperf with VMA
	  • Example Running sockperf Ping-pong Test
	  • Beta Level Features Configuration Parameters
	• Updated the following sections:
	  • VMA Configuration Parameters
	  • Configuration Parameter Values
	• Removed the Installation and Initial Configuration chapter. Was moved to the Installation Guide

Rev 6.3.28	Updated the following sections: • Target Applications • VMA Configuration Parameters • Configuration Parameter Values • Problem: Incorrect IGMP version • Problem: Lack of huge page resources in the system
Rev 6.1	• Updated sections in "Introduction to VMA" chapter and "VMA Library Architecture" for offload over InfiniBand • Updated "VMA System Requirements" section • Updated "Configuring the ConnectX-3 HCA for VMA" section • Removed deprecated parameters for NetEffect from "VMA Configuration Parameter Values" section

Version	Description
	• Updated Appendix on sockperf.
Rev 6.0	• Updated chapter on "VMA Library Architecture." • Updated graphic to reflect support of ConnectX3. • Removed outdated sections, "Unicast Support" and "Link and Port Recovery." • Updated sections in chapter on "Installation and Initial Configuration." • VMA System Requirements • All sections on VMA Installation and Upgrade • Updated chapter on "Configuring VMA." • Removed information about configuring a virtual MAC interface for unicast offload (deprecated feature). • Updated information for VMA configuration parameters VMA_THREAD_MODE, VMA_CLOSE_ON_DUP2, VMA_CONFIG_FILE, VMA_APPLICATION_ID. • Updated default values for VMA configuration parameters VMA_RX_POLL and VMA_SELECT_POLL. • Updated configuration parameter descriptions to include poll(). • Removed chapter "Tuning VMA". This information will be included in a separate Performance Tuning Guide, which will be part of the VMA 6.0 Documentation package. • Updated information for "Zero-copy revcfrom." • Added troubleshooting topic for "UMCAST enabled".

About This Manual

Audience

This manual is primarily intended for:

- 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.

Related Documentation

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

- Mellanox Messaging Accelerator (VMA) Library for Linux Release Notes (DOC-00329)
- Mellanox Messaging Accelerator (VMA) Installation Guide (DOC-10055)
- Performance Tuning Guidelines for Mellanox Network Adapters (DOC 3368)

Document Conventions

NOTE: Identifies important information that contains helpful suggestions.

CAUTION: Alerts you to risk of personal injury, system damage, or loss of data.

WARNING: Warns you that failure to take or avoid a specific action might result in personal injury or a malfunction of the hardware or software. Be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents before you work on any equipment.

Typography

The following table describes typographical conventions in Mellanox documentation. 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.
Mellanox CLI Guest Mode	`Switch >`	Mellanox CLI Guest Mode.
Mellanox CLI admin mode	`Switch #`	Mellanox CLI admin mode
String	`< > or []`	Strings in < > or [ ] are descriptions of what will actually be shown on the screen, for example, the contents of <your ip> could be 192.168.1.1
Management GUI label, item name	**New Network, New Environment**	Management GUI labels and item names appear in bold, whether or not the name is explicitly displayed (for example, buttons and icons).
User text entered into Manager, e.g., to assign as the name of a logical object	**"Env1", "Network1"**	Note the quotes. The text entered does not include the quotes.

1 Introduction to VMA

1.1 VMA Overview

The Mellanox 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 InfiniBand or 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 enhances 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.

1.2 Basic Features

The VMA library utilizes the direct hardware access and advanced polling techniques of RDMA-capable network cards. Utilization of InfiniBand's and 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
  - Higher throughput
  - 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.

1.3 Target Applications

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

- Fast transaction-based network applications, which require a high rate of request-response type operations over TCP or UDP unicast. This also includes any send/receive to/from an external network entity, such as a Market Data Order Gateway application working with an exchange.
- Market-data feed-handler software which consumes multicast data feeds (and which often use multicast as a distribution mechanism downstream), such as Wombat WDF and Reuters RMDS, or any home-grown feed handlers.
- Messaging applications responsible for producing/consuming relatively large amounts of multicast data including applications that use messaging middleware, such as Tibco Rendezvous (RV).

- Caching/data distribution applications, which utilize quick network transactions for cache creation/state maintenance, such as MemCacheD and Redis.
- Applications that handle distributed denial of service (DDoS) and web services applications with a heavy load of DNS requests.
- Messaging applications, such as UMS Informatica, which VMA 6.4 was certified with
- 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.

1.4 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 InfiniBand or 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 or IPoIB 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

2 VMA Library Architecture

2.1 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*.

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 usual 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 InfiniBand HCA or 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 InfiniBand HCA or 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.

2.2 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) or Stream Control Transmission Protocol (SCTP).

3 Installing VMA

For detailed information on how to install the VMA software, please refer to the VMA Installation Guide.

4 Configuring VMA

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

4.1 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 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 VMA to offload everything.

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 (on page 15).

4.1.1 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 subexpressions 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| *>
```

Table 2: Target Process Statement Options

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*.

4.1.2 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 subexpressions 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|*>
```

Table 3: Socket Transport Statement Options

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

Option	Description
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]` where port numbers are > 0 and < 65536

4.1.3 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
```

Note: You must also set the VMA parameter:

```
VMA_APPLICATION_ID=B1
```

4.2 VMA Configuration Parameters

The VMA configuration parameters are Linux OS environment variables, and 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.

Example:

```
VMA INFO : ------------------------------------------------------------------
----------
VMA INFO : VMA_VERSION: 6.9.1-0 Release built on 2015-05-10-16:20:19
VMA INFO : Cmd Line: sockperf sr
VMA DEBUG : Current Time: Sun May 10 17:35:35 2015
VMA DEBUG : Pid: 24714
VMA DEBUG : OFED Version: MLNX_OFED_LINUX-2.3-0.2.5:
VMA DEBUG : System: 2.6.32-220.el6.x86_64
VMA DEBUG : Architecture: x86_64
VMA DEBUG : Node: hail16
VMA DEBUG : ------------------------------------------------------------------
-----------
Log Level                       4            [VMA_TRACELEVEL]
Log Details                     2            [VMA_LOG_DETAILS]
Log Colors                      Enabled      [VMA_LOG_COLORS]
Log File                                     [VMA_LOG_FILE]
Stats File                                   [VMA_STATS_FILE]
Stats shared memory directory   /tmp/        [VMA_STATS_SHMEM_DIR]
Stats FD Num (max)              100          [VMA_STATS_FD_NUM]
Conf File                       /etc/libvma.conf    [VMA_CONFIG_FILE]
Application ID VMA_DEFAULT_APPLICATION_ID    [VMA_APPLICATION_ID]
Polling CPU idle usage          Disabled     [VMA_CPU_USAGE_STATS]
SigIntr Ctrl-C Handle           Disabled     [VMA_HANDLE_SIGINTR]
SegFault Backtrace              Disabled     [VMA_HANDLE_SIGSEGV]
Ring allocation logic TX        0 (Ring per interface)
[VMA_RING_ALLOCATION_LOGIC_TX]
Ring allocation logic RX        0 (Ring per interface)
[VMA_RING_ALLOCATION_LOGIC_RX]
Ring migration ratio TX         100          [VMA_RING_MIGRATION_RATIO_TX]
Ring migration ratio RX         100          [VMA_RING_MIGRATION_RATIO_RX]
Ring limit per interface        0 (no limit) [VMA_RING_LIMIT_PER_INTERFACE]
TCP max syn-fin rate            0 (no limit) [VMA_TCP_MAX_SYN_FIN_RATE]
Tx Mem Bufs TCP                 1000000      [VMA_TX_SEGS_TCP]
Tx Mem Bufs                     200000             [VMA_TX_BUFS]
Tx QP WRE                       16000        [VMA_TX_WRE]
Tx Max QP INLINE                220          [VMA_TX_MAX_INLINE]
Tx MC Loopback                  Enabled      [VMA_TX_MC_LOOPBACK]
Tx non-blocked eagains          Disabled     [VMA_TX_NONBLOCKED_EAGAINS]
Tx Prefetch Bytes               256          [VMA_TX_PREFETCH_BYTES]
Tx backlog max                  100          [VMA_TX_BACKLOG_MAX]
Rx Mem Bufs                     200000       [VMA_RX_BUFS]
Rx QP WRE                       16000        [VMA_RX_WRE]
Rx QP WRE BATCHING              64           [VMA_RX_WRE_BATCHING]
Rx Byte Min Limit               65536        [VMA_RX_BYTES_MIN]
Rx Poll Loops                   100000       [VMA_RX_POLL]
Rx Poll Init Loops              0            [VMA_RX_POLL_INIT]
Rx UDP Poll OS Ratio            100          [VMA_RX_UDP_POLL_OS_RATIO]
Rx Poll Yield                   Disabled     [VMA_RX_POLL_YIELD]
Rx Prefetch Bytes               256          [VMA_RX_PREFETCH_BYTES]
Rx Prefetch Bytes Before Poll   0        [VMA_RX_PREFETCH_BYTES_BEFORE_POLL]
Rx CQ Drain Rate                Disabled     [VMA_RX_CQ_DRAIN_RATE_NSEC]
GRO max streams                 32           [VMA_GRO_STREAMS_MAX]
TCP 3T rules                    Disabled      [VMA_TCP_3T_RULES]
ETH MC L2 only rules            Disabled      [VMA_ETH_MC_L2_ONLY_RULES]
Select Poll (usec)              100000       [VMA_SELECT_POLL]
Select Poll OS Force            Disabled     [VMA_SELECT_POLL_OS_FORCE]
Select Poll OS Ratio            10           [VMA_SELECT_POLL_OS_RATIO]
Select Poll Yield               Disabled     [VMA_SELECT_POLL_YIELD]
Select Skip OS                  4            [VMA_SELECT_SKIP_OS]
```

```
Select CQ Interrupts             Enabled       [VMA_SELECT_CQ_IRQ]
CQ Drain Interval (msec)         10            [VMA_PROGRESS_ENGINE_INTERVAL]
CQ Drain WCE (max)               10000         [VMA_PROGRESS_ENGINE_WCE_MAX]
CQ Interrupts Moderation         Enabled       [VMA_CQ_MODERATION_ENABLE]
CQ Moderation Count              48            [VMA_CQ_MODERATION_COUNT]
CQ Moderation Period (usec)      50            [VMA_CQ_MODERATION_PERIOD_USEC]
CQ AIM Max Count                 560           [VMA_CQ_AIM_MAX_COUNT]
CQ AIM Max Period (usec)         250           [VMA_CQ_AIM_MAX_PERIOD_USEC]
CQ AIM Interval (msec)           250           [VMA_CQ_AIM_INTERVAL_MSEC]
CQ AIM Interrupts Rate (per sec) 5000
[VMA_CQ_AIM_INTERRUPTS_RATE_PER_SEC]
CQ Poll Batch (max)              16            [VMA_CQ_POLL_BATCH_MAX]
CQ Keeps QP Full                 Enabled       [VMA_CQ_KEEP_QP_FULL]
QP Compensation Level            256           [VMA_QP_COMPENSATION_LEVEL]
Offloaded Sockets                Enabled       [VMA_OFFLOADED_SOCKETS]
Timer Resolution (msec)          10            [VMA_TIMER_RESOLUTION_MSEC]
TCP Timer Resolution (msec)      100           [VMA_TCP_TIMER_RESOLUTION_MSEC]
Delay after join (msec)          0             [VMA_WAIT_AFTER_JOIN_MSEC]
Delay after rereg (msec)         500           [VMA_WAIT_AFTER_REREG_MSEC]
Internal Thread Affinity         -1            [VMA_INTERNAL_THREAD_AFFINITY]
Internal Thread Cpuset                         [VMA_INTERNAL_THREAD_CPUSET]
Internal Thread Arm CQ Disabled                [VMA_INTERNAL_THREAD_ARM_CQ]
Thread mode Multi spin lock                    [VMA_THREAD_MODE]
Mem Allocate type                1 (Contig Pages) [VMA_MEM_ALLOC_TYPE]
Num of UC ARPs                   3             [VMA_NEIGH_UC_ARP_QUATA]
UC ARP delay (msec)              10000         [VMA_NEIGH_UC_ARP_DELAY_MSEC]
Num of neigh restart retries     1             [VMA_NEIGH_NUM_ERR_RETRIES]
IPOIB support                    Enabled       [VMA_IPOIB]
BF (Blue Flame)                  Enabled       [VMA_BF]
fork() support                   Enabled       [VMA_FORK]
close on dup2()                  Enabled       [VMA_CLOSE_ON_DUP2]
MTU                              1500          [VMA_MTU]
MSS                              0 (follow VMA_MTU) [VMA_MSS]
TCP CC Algorithm                 0 (LWIP)      [VMA_TCP_CC_ALGO]
TCP scaling window               3             [VMA_WINDOW_SCALING]Suppress
IGMP ver. warning  Disabled      [VMA_SUPPRESS_IGMP_WARNING]
```

4.2.1 Configuration Parameter Values

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

Table 4: Configuration Parameter Values

VMA Configuration Parameter	Description and Examples
VMA_TRACELEVEL	0 = PANIC – Panic level logging. This trace level causes fatal behavior and halts the application, typically, caused by memory allocation problems. PANIC level is rarely used.
	1 = ERROR – 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.
	2 = WARNING – Runtime warning that does not disrupt the application workflow. 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.

VMA Configuration Parameter	Description and Examples
	3 = INFO – 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.
	4 = DEBUG – 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.
	5 = FUNC – 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.
	6 = FUNC_ALL – 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_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.

VMA Configuration Parameter	Description and Examples
	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. In case this value is set to an empty string: " ", no shared memory files are created. Default: **/tmp/**
VMA_TCP_MAX_SYN_FIN_RATE	Limits the number of TCP control packets (TCP SYN/FIN/RST packets) that VMA handles per second for each thread. Example: by setting this value to 10, the maximal number of TCP control packets accepted by VMA per second for each thread will be 10. Set this value to 0 for VMA to handle an un-limited number of TCP control packets per second for each thread. 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 Configuration Parameter	Description and Examples
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: **16000** 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_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_TX_MAX_INLINE=0 disables INLINEing on the TX transmit path. In older releases this parameter was called VMA_MAX_INLINE. Default: **224**
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 bytes**
VMA_RX_WRE	The number of Work Request Elements allocated in all received QPs. Default: **16000**

VMA Configuration Parameter	Description and Examples
VMA_RX_BYTES_MIN	The minimum value in bytes used per socket by the VMA when applications call to *setsockopt(SO_RCVBUF)*. 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 (on page 37).
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_OS_RATIO	Defines the ratio between VMA CQ poll and OS FD poll. This will result in a single poll of the not-offloaded sockets every

VMA Configuration Parameter	Description and Examples
	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_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. 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_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 Configuration Parameter	Description and Examples
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 (on page 37).
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_POLL_YIELD	When an application runs with multiple threads on a limited number of cores, each thread polling inside VMA (`select()`, `poll()`, or `epoll_wait()`) should yield the CPU to other polling threads so as not to starve them from processing incoming packets. Default: **0** (Disabled)
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_SELECT_CQ_IRQ	When disabled, no InfiniBand interrupts are used during `select()`, `poll()`, or `epoll_wait()` socket calls. This mode of work is not recommended. This parameter is used by applications that use `VMA_SELECT_POLL` for polling (with the default zero millisecond timeout). Range: **0** - Disabled, **1** - Enabled Default: **1** (Enabled)
VMA_CQ_POLL_BATCH_MAX	The maximum size of the array while polling the CQs in the VMA. Default: **8**

VMA Configuration Parameter	Description and Examples
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. 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 micro-seconds 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. If enabled, CQ tries to compensate QP for each polled receive completion. If there is a shortage of buffers, it reposts a

VMA Configuration Parameter	Description and Examples
	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_THREAD_MODE`	By default VMA is ready for multi-threaded applications, meaning it is thread-safe. 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_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.

VMA Configuration Parameter	Description and Examples
	• 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_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 (Mellanox OFED 1.5.3 and later). VMA allocates huge pages (VMA_HUGETBL) by default. Default: **1** (Enabled)
VMA_MTU	Sets the fragmentation size of the packets sent by the VMA library. This value determines the size of each Rx and Tx buffer. Default: **1500** bytes Recommendations: • Set to**1500** for Ethernet networks or interoperability with Ethernet networks.
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_WINDOW_SCALING	TCP scaling window. This value (factor range from 0 to 14, -1 to disable, -2 to use OS value) sets the factor in which the TCP window is scaled. Factor of 0 allows using the TCP scaling window of the remote host, while not changing the window of the local host. Value of -1 disables both directions. Value of -2 uses the OS maximums and receives buffer value to calculate the factor. Make sure that VMA buffers are big enough to support the window. Default: **3**

VMA Configuration Parameter	Description and Examples
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_THREAD_AFFINITY	Controls which CPU core(s) the VMA internal thread is serviced on. The CPU set should be provided as either a hexidecmal value that represents 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_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 Default: **0** (milli-sec)
VMA_NEIGH_UC_ARP_QUATA	VMA will send UC ARP in case neigh state is NUD_STALE. In case that neigh state is still NUD_STALE VMA will try

VMA Configuration Parameter	Description and Examples
	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_SUPPRESS_IGMP_WARNING	Use VMA_SUPPRESS_IGMP_WARNING=1 to suppress the warnings about igmp version not forced to be 2. Default: **0** (Disabled)
VMA_BF	Enables/disables BlueFlame usage of the card. Default: **1** (Enabled)

4.2.2 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 Multi-thread applications.

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

Table 5: Beta Level Configuration Parameter Values

VMA Configuration Parameter	Description and Examples
VMA_RING_ALLOCATION_LOGIC_TX VMA_RING_ALLOCATION_LOGIC_RX	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. The logic options are: 0 - Ring per interface 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	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.

VMA Configuration Parameter	Description and Examples
VMA_RING_MIGRATION_RATIO_RX	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 ring consumes VMA_RX_WRE buffers.] Use a value of 0 for an unlimited number of rings. Default: **0** (no limit)
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. Default: **0** (LWIP).

5 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*.

- ***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
```

6 Example Running sockperf Ping-pong Test

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 1.1.1.12
```

Client expected output:

```
$LD_PRELOAD=libvma.so sockperf pp -i 1.1.1.12
VMA INFO   : --------------------------------------------------------------
---------------
 VMA INFO   : VMA_VERSION: 6.8.2-0 Release built on 2013-12-11-16:20:19
 VMA INFO   : Cmd Line: sockperf pp -i 1.1.1.12
 VMA INFO   : Log Level                     3
[VMA_TRACELEVEL]
 VMA INFO   : --------------------------------------------------------------
---------------
mlx4: prefer_bf=1
mlx4: prefer_bf=1
sockperf: == version #2.5.231 ==
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on socket(s)

[ 0] IP = 2.2.2.15        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=5.100 sec; SentMessages=2240397;
ReceivedMessages=2240396
sockperf: ========= Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=4.988 sec; SentMessages=2218152;
ReceivedMessages=2218152
sockperf: ====> avg-lat=  1.108 (std-dev=0.244)
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-
order messages = 0
sockperf: Summary: Latency is 1.108 usec
sockperf: Total 2218152 observations; each percentile contains 22181.52
observations
sockperf: ---> <MAX> observation =   19.023
sockperf: ---> percentile  99.99 =    6.347
sockperf: ---> percentile  99.90 =    3.559
sockperf: ---> percentile  99.50 =    2.302
sockperf: ---> percentile  99.00 =    2.071
sockperf: ---> percentile  95.00 =    1.539
sockperf: ---> percentile  90.00 =    1.150
sockperf: ---> percentile  75.00 =    1.085
sockperf: ---> percentile  50.00 =    1.050
sockperf: ---> percentile  25.00 =    1.022
sockperf: ---> <MIN> observation =    0.979
```

3. Analyze the client output:

```
Average latency: 1.108 usec
```

7 VMA Extra API

7.1 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.`

7.2 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.

7.3 Control Off-load Capabilities During Run-Time

7.3.1 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

Table 6: add_conf_rule Parameters

Parameter Name	Description	Values
Config_line	New rule to add to the top of the list (highest priority).	A char buffer with the exact format as defined in libvma.conf, and should end with '\0'

7.3.2 Creating Sockets as Off-loaded or Not-Off-loaded

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

Table 7: add_conf_rule Parameters

Parameter Name	Description	Values
offload	Offload property	1 for offloaded, 0 for not-offloaded
tid	Thread ID	

7.4 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()`.

Table 8: Packet Filtering Callback Function Parameters

Parameter Name	Description	Values
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.	
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 `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.

7.4.1 Zero Copy recvfrom()

Description: 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);`

Parameters:

Table 9: Zero-copy revcfrom Parameters

Parameter Name	Description	Values
s	Socket file descriptor	

Parameter Name	Description	Values
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** argument 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()`).

Return Values:

If the return value is positive, data copy has been performed.

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

7.4.2 Freeing Zero Copied Packet Buffers

Description: 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);
```

Parameters:

Table 10: Freeing Zero-copy Datagram Parameters

Parameter Name	Description	Values
s	Socket from which the packet was received.	
pkts	Array of packet identifiers.	
count	Number of packets in the array.	

Return Values:

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.
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.

8 Debugging, Troubleshooting, and Monitoring

8.1 Monitoring – the vma_stats Utility

Networking applications open various types of sockets.

The VMA library holds the following counters:

- Separate performance counters for each socket of the datagram (UDP) IP family type.
- 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

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] [-v view] [-d details] [-i interval]
```

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

Table 11: vma_stats Utility Options

Parameter Name	Argument	Parameter Description and Values
`-p, --pid`	`<pid>`	Shows VMA statistics for a process with pid: <pid>.
`-v, --view`	`<1|2|3|4>`	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.
`-d, --details`	`<1|2>`	Sets the details mode: 1. Show totals (default). 2. Show deltas.
`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**

Parameter Name	Argument	Parameter Description and Values
--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.
-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.

8.1.1 Examples

The following sections contain examples of the vma_stats utility.

8.1.1.1 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
-------------------------------------------------------------------------
```

Analysis of the Output:

- 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.

8.1.1.2 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%
```

Analysis of the Output:

- Three sockets were created (fds: 15, 19, and 23).
- Received 11590 packets, 22 Kilobytes during the last second via fds: 15 and 19.
- Transmitted 11590 packets, 22 Kbytes during the last second via fds: 15 and 19.
- Not all the traffic was offloaded, as fd 23: 11590 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 reamining 30% is used for processing the data by the application.

8.1.1.3 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 KB / 786133 / 0 / 0 [bytes/packets/eagains/errors]/s
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 /s
Packets queue len:            0
Drained max:                511
Buffer pool size:           500
Buffer disorder:           0.01%
=======================================================
=======================================================
        RING=[0]
Packets count:           786133 /s
Packets bytes:       1192953545 /s
Interrupt requests:      786137 /s
Interrupt received:       78613 /s
Moderation frame count:          10
Moderation usec period:         181
=======================================================
```

Analysis of the Output:

- 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 by the CQ and are waiting to be processed by the upper layers)
- The maximum number of packets drained by the CQ during a single drain cycle is 511 (see VMA_CQ_DRAIN_WCE_MAX)
- The RING received 786133 packets during this period
- The RING received 1192953545 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

8.1.1.4 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
```

If the user application performed transmit or receive activity on a socket, those values will be logged when the sockets are closed. The VMA logs its internal performance counters if VMA_TRACELEVEL=4 (see Example 5).

8.1.1.5 Example 5

Description: This is an example of a log of socket performance counters along with an explanation of the results.

Output:

```
VMA: [fd=10] Tx Offload: 455 KB / 233020 / 0 [bytes/packets/errors]
VMA: [fd=10] Tx OS info: 0 KB / 0 / 0 [bytes/packets/errors]
VMA: [fd=10] Rx Offload: 455 KB / 233020 / 0 [bytes/packets/errors]
VMA: [fd=10] Rx OS info: 0 KB / 0 / 0 [bytes/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]
```

Analysis of the Output:

- 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.

8.2 Debugging

8.2.1 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 (on page 15)) 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. Use the Troubleshooting (on page 42) 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.

8.2.2 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).

8.2.3 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/
```

8.3 Troubleshooting

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

- **Problem:** 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**.

- **Problem:** 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).

- **Problem:** 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).

- **Problem:** 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.

- **Problem:** Incorrect IGMP version

The following warning is reported:

```
VMA: WARNING: ***************************************************
VMA: WARNING: IGMP Version flag is not forced to IGMPv2 for interface ib2
while
VMA: WARNING: VMA_IGMP is Enabled!
VMA: WARNING: Working in this mode can cause issues due to Eth-IB gateway
requirements
VMA: WARNING: Please "echo 2 >
/proc/sys/net/ipv4/conf/ib2/force_igmp_version"
VMA: WARNING: or "export VMA_IGMP=0"
VMA: WARNING: before loading your application with VMA library
VMA: WARNING: Read the IGMP section in the VMA's User Manual for more
information
VMA: WARNING: ***************************************************
```

This warning message means that you are using IGMP version other than 2, which is the version supported by VMA. (Version 2 is required for the Eth-IB gateway.)

Solution: Use VMA_SUPPRESS_IGMP_WARNING=1 if you are working in an InfiniBand fabric and do not need to receive multicast packets from the Ethernet to the InfiniBand fabric or you are working in an Ethernet fabric.

If you do expect to receive multicast packets from the Ethernet to the InfiniBand fabric with VMA, force IGMP working mode to version 2 in all your hosts, as well as in your routers:

```
echo 2 > /proc/sys/net/ipv4/conf/ib2/force_igmp_version
```

- **Problem:** UMCAST is enabled

The following warning is reported:

```
VMA WARNING: ****************************************************
VMA WARNING: UMCAST flag is Enabled for interface ib0!
VMA WARNING: Please disable it: "echo 0 > /sys/class/net/ib0/umcast"
VMA WARNING: This option in no longer needed in this version
VMA WARNING: Please refer to Release Notes for more information
VMA WARNING: ****************************************************
```

This warning message means that the UMCAST flag is on.

Solution: Turn off the UMCAST flag. This option is no longer needed in this version.

- **Problem:** 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: 1. Disable VMA's hugepage support
(VMA_HUGETLB=0)*
VMA: INFO   : *          2. Restart process after increasing the number
of*
VMA: INFO   : *             hugepages resources in the system: *
VMA: INFO   : * "cat /proc/meminfo |  grep -i HugePage"
*
VMA: INFO   : * "echo 1000000000 > /proc/sys/kernel/shmmax"
*
VMA: INFO   : * "echo 400 > /proc/sys/vm/nr_hugepages"
*
VMA: WARNING: * Read more about the Huge Pages in the VMA User Manual
*
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

Solution: Set VMA_MEM_ALLOC_TYPE= 1, in order to enable VMA's contig pages alloc**ation log**ic, this is the default setting;

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 400 > /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.

CAUTION: 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;
```

Appendix A: 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.

A.1 Overview

Sockperf is an open source utility. For more general information, see http://code.google.com/p/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 over 1 Gb and 10Gb Ethernet, see:

- Latency with Ping-pong Test (on page 48)
- Bandwidth and Packet Rate With Throughput Test (on page 50)

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 (on page 47)).

A.1.1 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.

A.2 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 10 Gb 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).

A.3 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.

A.3.1 UDP MC Ping-pong Over 1 Gb

➢ ***To run UDP MC ping-pong over 1 Gb Ethernet***

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

```
# route add -net 224.0.0.0 netmask 240.0.0.0 dev eth1
```

where *eth1* is the 1 Gb Ethernet interface.

5. Run the server by using:

```
# sockperf sr -i <server-1g-ip>
```

6. Run the client by using:

```
# sockperf pp -i <server-1g-ip>
```

The following output is obtained:

```
sockperf: Warmup stage (sending a few dummy packets)...
sockperf: Starting test...
sockperf: Test end (interrupted by timer)
sockperf: [Total Run] RunTime=1.100 sec; SentMessages=36304;
ReceivedMessages=36303
sockperf: ========= Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=1.000 sec; SentMessages=33026;
ReceivedMessages=33026
sockperf: ====> avg-lat= 15.096 (std-dev=0.300)
sockperf: # dropped packets = 0; # duplicated packets = 0; # out-of-order
packets = 0
sockperf: Summary: Latency is 15.096 usec
sockperf: Total 33026 observations; each percentile contains 330.26
observations
sockperf: ---> <MAX> observation = 36.855
sockperf: ---> percentile 99.99 = 25.553
sockperf: ---> percentile 99.00 = 15.803
sockperf: ---> percentile 50.00 = 15.080
sockperf: ---> <MIN> observation = 13.406
```

Interpretation of the results:

The example shows an average latency of 15.096 usec

A.3.2 UDP MC Ping-pong Over 10 Gb

➢ ***To run UDP MC ping-pong over 10 Gb Ethernet***

7. After configuring the routing table as described in Configuring the Routing Table for Multicast Tests (on page 47), run the server by using:

```
# sockperf sr -i <server-10g-ip>
```

8. Run the client by using:

```
# sockperf pp -i <server-10g-ip>
```

The following output is obtained:

```
sockperf: [Total Run] RunTime=1.100 sec; SentMessages=79960;
ReceivedMessages=79959
sockperf: [Valid Duration] RunTime=1.000 sec; SentMessages=72803;
ReceivedMessages=72803
sockperf: ====> avg-lat=  6.825 (std-dev=0.261)
sockperf: Summary: Latency is 6.825 usec
sockperf: Total 72803 observations; each percentile contains 728.03
observations
sockperf: ---> <MAX> observation =   19.057
sockperf: ---> percentile  99.99 =    9.159
sockperf: ---> percentile  99.00 =    7.382
sockperf: ---> percentile  50.00 =    6.830
sockperf: ---> <MIN> observation =    5.380
```

Interpretation of the results:

The example shows an average latency of 6.825 usec

A.3.3 UDP MC Ping-pong Over 10 Gb + VMA

- ***To run UDP MC ping-pong over 10 Gb Ethernet + VMA***

9. After configuring the routing table as described in Configuring the Routing Table for Multicast Tests (on page 47), run the server by using:

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

10. Run the client by using:

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

The following output is obtained:

```
VMA INFO   : ------------------------------------------------
 VMA INFO   : Current Time: Sun Jan 29 13:32:40 2012
 VMA INFO   : Cmd Line: sockperf pp -i 224.4.2.216
 VMA INFO   : Pid:  4215
 VMA INFO   : OFED Version: OFED-VMA-1.5.3-0006:
 VMA INFO   : System: 2.6.32-71.el6.x86_64
 VMA INFO   : Architecture: x86_64
 VMA INFO   : Node: boo2
 VMA INFO   : ------------------------------------------------------------
 VMA INFO   :  Log Level               3               [VMA_TRACELEVEL]
 VMA INFO   :  Log File                                [VMA_LOG_FILE]
 VMA INFO   : ------------------------------------------------------------
 VMA INFO   :
************************************************************
 VMA INFO   :
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on socket(s)

[ 0] IP = 224.4.2.216     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=1.100 sec; SentMessages=299903;
ReceivedMessages=299902
sockperf: ========= Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=1.000 sec; SentMessages=272956;
ReceivedMessages=272956
sockperf: ====> avg-lat=  1.809 (std-dev=0.244)
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-order
messages = 0
sockperf: Summary: Latency is 1.809 usec
```

```
sockperf: Total 272956 observations; each percentile contains 2729.56
observations
sockperf: ---> <MAX> observation =    7.489
sockperf: ---> percentile  99.99 =    3.897
sockperf: ---> percentile  99.00 =    2.850
sockperf: ---> percentile  50.00 =    1.717
sockperf: ---> <MIN> observation =    1.579
```

Interpretation of the results:

The example shows an average latency of 1.809 usec

A.3.4 UDP MC Ping-pong Summary

Table 12: UDP MC Ping-pong Results

Test	1 Gb Ethernet	10 Gb Ethernet	10 Gb Ethernet + VMA
Latency	15.096 usec	6.825 usec	1.809 usec
VMA Improvement	13.287 usec (89%)	5.016 usec (73%)	

A.4 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.

A.4.1 TCP Throughput Over 10 Gb

- ***To run TCP throughput over 10 Gb***

11. Run the server by using:

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

12. Run the client by using:

```
# sockperf tp --tcp -i <server-ip> -m 100
```

where *-m/--msg-size* is the minimum message size in bytes (minimum default 12).

The following output is obtained:

```
sockperf: Total of 1282013 messages sent in 1.100 sec
sockperf: Summary: Message Rate is 1165457 [msg/sec]
sockperf: Summary: BandWidth is 13.338 MBps (106.701 Mbps)
```

Notes:

- You can use `--tcp-avoid-nodelay` to deliver TCP messages immediately (default ON).
- For more sockperf throughput options run:

```
# sockperf tp -h
```

A.4.2 TCP Throughput Over 10 Gb+VMA

- ***To run TCP throughput over 10 Gb + VMA***

13. Run the server by using:

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

14. Run the client by using:

```
# LD_PRELOAD=libvma.so sockperf tp --tcp -i <server-ip> -m100
```

The following output is obtained:

```
sockperf: Total of 8413778 messages sent in 1.100 sec
sockperf: Summary: Message Rate is 7648873 [msg/sec]
sockperf: Summary: BandWidth is 87.534 MBps (700.275 Mbps)
```

A.4.3 TCP Throughput Summary

Table 13: TCP Throughput Results

Test	10 Gb Ethernet	10 Gb Ethernet + VMA
Message Rate	1165457 [msg/sec]	7648873 [msg/sec]
Bandwidth	13.338 MBps (106.701 Mbps)	87.534 MBps (700.275 Mbps)
VMA Improvement	74.196 (562%)	

A.5 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
```

Table 14: Available Subcommands

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 http://code.google.com/p/sockperf/.

A.5.1 Additional Options

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

Table 15: General sockperf 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	Tread multiple ip+port combinations from file <file> (server uses select).
-F	--iomux-type	Type of multiple file descriptors handle [s|select|p|poll|e|epoll|r|recvfrom] (default select).
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 delivering TCP Messages Immediately (default ON).
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	--buffer-size	Set total socket receive/send buffer <size> in bytes (system defined by default).
N/A	--vmazcopyread	If possible use VMA's zero copy reads API (see the VMA readme).
N/A	--daemonize	Run as daemon.
N/A	--nonblocked	Open non-blocked sockets.
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	--no-rdtsc	Do not use the register when measuring time; instead use the monotonic clock.
N/A	--set-sock-accl	Set socket acceleration before running (available for some Mellanox systems).
N/A	--load-vma	Load VMA dynamically even when LD_PRELOAD was not used.

Short Command	Full Command	Description
N/A	--tcp-skip-blocking-send	Enables non-blocking send operation (default OFF).
N/A	--recv_looping_num	Set sockperf to loop over recvfrom() until EAGAIN or <N> good received packets, -1 for infinite, must be used with --nonblocked (default 1).
-d	--debug	Print extra debug information.

Table 16: Client Options

Short Command	Full Command	Description
N/A	--srv-num	Set the number of servers the client works with.
N/A	--sender-affinity	Set sender thread affinity to the given core IDs in the list format (see: cat /proc/cpuinfo).
N/A	--receiver-affinity	Set receiver thread affinity to the given core IDs in the list format (see: cat /proc/cpuinfo).
N/A	--full-log	Dump full log of all message send/receive times to the given file in CSV format.
-t	--time	Set the number of seconds to run (default 1, max = 36000000).
-b	--burst	Control the number of messages sent from the client in every burst.
N/A	--giga-size	Print sizes in GigaBytes.
N/A	--increase_output_precision	Increase number of digits after the decimal point of the throughput output (from 3 to 9).
N/A	--mps	Set number of messages-per-second (default = 10000 for under-load mode, or max for ping-pong and throughput modes); for maximum use --mps=max. (Supports --pps for backward compatibility.)
-m	--msg-size	Use messages of minimum size in bytes (minimum default 12 bytes).
-r	--range	Use with -m to randomly change the minimum message size in range: <size> +- <N>.

Table 17: 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).

Short Command	Full Command	Description
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.
-m	--msg-size	Set maximum message size that the server can receive <size> bytes (default 65506).
-g	--gap-detection	Enable gap-detection.

A.5.2 Sending Bursts

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

A.6 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.

A.7 Troubleshooting sockperf

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.

Appendix B: Multicast Routing

B.1 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_MEMBERSHP` 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.

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 InfiniBand/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 C: Acronyms

Table 18: Acronym Table

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	Mellanox Messaging Accelerator
VMS	VMA Messaging Service
WCE	Work Completion Elements