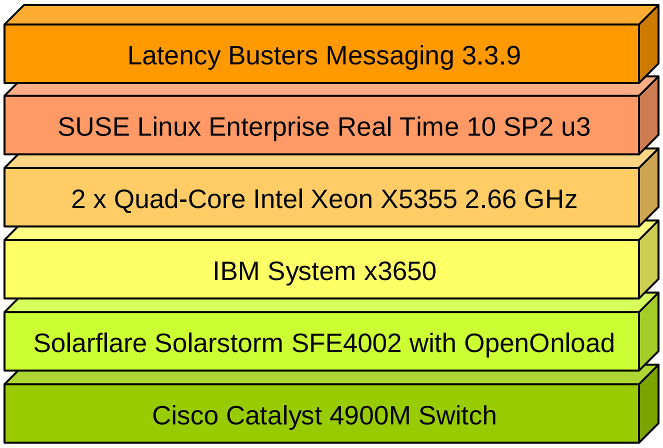
Latency Busters Messaging 3.3.9 with Cisco Catalyst 4900M 10GigE Switch and Solarflare NIC with OpenOnload

Issue 1.0, 18 May 09

Technology Stack Under Test



Key Results

➔ One-way, send-to-end latency of a latency-optimized configuration with OpenOnload, at up to 125K messages/second (mps):
- Mean did not exceed 19 microseconds
- 99.9th percentile did not exceed 63 microseconds
- Standard deviation did not exceed 6 microseconds

➔ Throughput of 64-byte messages with 3 publishers using UDP from a single server in a throughput-optimized configuration:
- 1.86M mps / 0.95 gigabit per second (Gbps) with OpenOnload
- 2.46M mps / 1.26 Gbps with kernel stack

➔ Throughput of 1024-byte messages with 3 publishers using UDP from a single server in a different throughput-optimized configuration:
- 550K mps / 4.51 Gbps with OpenOnload
- 520K mps / 4.26 Gbps with kernel stack

NOTE: The tests in this STAC Report are not based on STAC Benchmark specifications because they are currently under development for this kind of workload by the STAC Benchmark Council. For more information, see www.STACresearch.com/council.

STAC SECURITIES TECHNOLOGY ANALYSIS CENTER

Disclaimer

The Securities Technology Analysis Center, LLC (STAC®) prepared this report at the request of Cisco Systems. It is provided for your internal use only and may not be redistributed, retransmitted, or published in any form without the prior written consent of STAC. All trademarks in this document belong to their respective owners.

The test results contained in this report are made available for informational purposes only. STAC does not guarantee similar performance results. All information contained herein is provided on an "AS-IS" BASIS WITHOUT WARRANTY OF ANY KIND. STAC has made commercially reasonable efforts to adhere to any vendor published test procedures and otherwise ensure the accuracy of the contents of this document, but the document may contain errors. STAC explicitly disclaims any liability whatsoever for any errors or otherwise.

The evaluations described in this document were conducted under controlled laboratory conditions. Obtaining repeatable, measurable performance results requires a controlled environment with specific hardware, software, network, and configuration in an isolated system. Adjusting any single element may yield different results. Additionally, test results at the component level may not be indicative of system level performance, or vice versa. Each organization has unique requirements and therefore may find this information insufficient for its needs.

Customers interested in a custom analysis for their environment are encouraged to contact STAC.

Contents

Summary

The rapid growth of data traffic in the capital markets continues to be a major concern for industry technologists. As markets become more volatile, huge volumes of traffic can overwhelm systems, increase latency unpredictably, and throw off application algorithms. In fact, some algorithmic trading applications are more sensitive to the predictability of latency than they are to the average latency (within limits).

Cisco believes that 10 Gigabit Ethernet (10GigE) will become the foundation of messaging systems used in the financial markets. Cisco asked STAC to measure the performance of 29West's Latency Busters® Messaging (LBM) middleware using Solarflare® Solarstorm™ SFE4002 10 Gigabit NICs with OpenOnload™ and a Cisco® Catalyst® 4900M 10 Gigabit switch. The platform used servers and processors commonly deployed at financial firms today. Several components of this solution stack (including LBM, the Solarflare stack, and the operating system) allowed the vendors to tune performance either to minimize latency or to maximize throughput. The goals of this project were to:

- Measure one-way message latency of a latency-optimized configuration sending messages from a single UDP publisher, then three UDP publishers on the same server, using payloads that range from 64 bytes to 1024 bytes. Do this at the maximum throughput that this configuration could support before maximum latency exceeded 1 millisecond when carrying 1024-byte payloads.

- Measure both maximum throughput and latency of a throughput-optimized configuration sending from a single UDP publisher, using payloads that range from 64 bytes to 1024 bytes. Then run three UDP publishers on the same server using both 64-byte and 1024-byte messages.

To summarize, we found:

- One-way, send-to-end latency of a latency-optimized configuration with OpenOnload, at up to 125K messages/ second (mps):
  - Mean did not exceed 19 microseconds
  - 99.9th percentile did not exceed 63 microseconds
  - Standard deviation did not exceed 6 microseconds

- Throughput of 64-byte messages with 3 publishers using UDP from a single server in a throughput-optimized configuration:
  - 1.86M mps / 0.95 gigabit per second (Gbps) with OpenOnload
  - 2.46M mps / 1.26 Gbps with kernel stack

- Throughput of 1024-byte messages with 3 publishers using UDP from a single server in a different throughput-optimized configuration:
  - 550K mps / 4.51 Gbps with OpenOnload
  - 520K mps / 4.26 Gbps with kernel stack

1. Background

Despite the financial crisis, firms that are active in liquid electronic markets continue to invest in the infrastructure that carries information into and out of the firm. For high-frequency traders, latency is a paramount concern. In some markets, firms can profit from less than one millisecond of advantage over competitors, which drives them to search for sub-millisecond optimizations throughout the systems fueling their trades. The latency obsession has resulted from the spread of automated trading across geographies and asset classes, and the resulting imperative to exploit—or defend against—new latency arbitrage opportunities.

Another consequence of automated trading is a ballooning of market data traffic volumes, which complicates the latency race, thanks to a well-established tradeoff between throughput and latency. Update-rate increases of 2 to 6 times in a single year are not uncommon for today's exchanges. Automated trading drives this traffic by both increasing transaction volumes and increasing the ratio of quotes and cancellations to actual trades. On top of this, large trading firms often generate enormous amounts of real-time data internally, which they pump onto their internal messaging systems.

This combination of forces keeps market data technologists on the lookout for new technologies that can shift the performance tradeoffs in the right direction. One layer of the technology stack that receives ongoing scrutiny is messaging, i.e., the transmission of information from one process to another, particularly across machine boundaries—i.e., over networks. An inefficient messaging stack can contribute significant latency or, worse, buckle under the pressure of traffic spikes. In cases where two alternate stacks deliver mean latency well below the trading firm's requirement, firms will often prefer the stack that minimizes latency jitter, since unpredictability of latency is something that trading algorithms find it difficult to deal with.

Cisco believes that 10 Gigabit Ethernet will become the foundation of messaging systems used in the financial markets. Cisco manufactures the Catalyst line of switches used today by many Wall Street firms. They believe that the Cisco Catalyst 4900M 10 Gigabit switch is an ideal upgrade path for firms looking for a low-latency Ethernet switch.

The Cisco Catalyst 4900M was designed to be a high-speed, low-latency, semi-modular, enterprise-quality L2/L3 data center switch. It can provide both 10/100/1000 and 10 Gigabit Ethernet ports. This provides firms with a migration path between multiple speeds and standards.

To demonstrate how well Cisco 10 Gigabit Ethernet works at accelerating messaging systems, Cisco chose 29West Latency Busters Messaging (LBM) messaging middleware (using the LBM-RM reliable UDP multicast transport), SolarFlare 10 Gigabit Ethernet NICs, and Novell's SUSE Linux Enterprise Real Time (SLERT).

29West designed LBM to be fast, lightweight, and efficient for delivering high message rates at very low-latency. LBM has many configuration parameters allowing developers and administrators to configure it optimally for the application's needs. 29West claims that LBM is widely used in financial services by firms seeking low-latency access to data.

Solarflare Communications provided their Solarstorm SFE4002 10 Gigabit Ethernet NIC. The Solarstorm™ 10GbE PCI-Express server adapter was designed to support virtualization and iSCSI protocol acceleration with line-rate performance and low power consumption, just 7 Watts. Solarflare claims that in both kernel and OpenOnload modes, the adapter supports financial services and HPC applications which demand very low latency and very high throughput.

Tests were performed using the standard kernel IP stack as well as SolarFlare OpenOnload technology. OpenOnload is an open-source high-performance network stack for Linux created by SolarFlare Communications. OpenOnload performs network processing at user-level and is binary-compatible with existing applications that use TCP/UDP with BSD sockets. It comprises a user-level shared library that implements the protocol stack, and a supporting kernel module. Applications can choose at runtime to use OpenOnload or the standard Linux kernel stack.

The servers used were IBM System x3650 2U rack-mounted servers with dual Quad-core Intel Xeon X5355 ("Clovertown") processors. Although the processors are two generations old, they were selected because they are believed to be in use at many financial firms today. The results from these tests are therefore characteristic of what firms can expect on existing hardware. Each server was loaded with SLERT, a real time operating system fully supported by Novell. SLERT was designed to reduce the latency and increase the predictability of time-sensitive mission-critical applications. Several of the capabilities unique to SLERT were used to tune the system performance.

Many of the components of this stack lend themselves to tuning for the performance characteristics most desirable for a given workload. For example, they can be optimized to minimize latency or to maximize throughput. STAC benchmarked the performance of this stack in both latency-optimized and throughput-optimized configurations at message sizes similar to those found in market data applications and internal publishing applications in the financial markets. For each configuration, we measured the one-way (i.e., publisher-to-subscriber) latency as well as the maximum throughput from a single server. For the latency-optimized configurations, maximum throughput was defined as the highest message rate for which maximum latency did not exceed one millisecond, while for throughput-optimized configurations, it was the highest sustainable message rate without LBM reporting data loss to the application.

2. Description of Tests

2.1 Methodology

2.1.1 Overview

We performed four types of tests designed to emulate a range of pub/sub use cases. Our goal was to show the latency and throughput characteristics of the stack under test (SUT) where we varied the number of publisher and consumer applications, the message rates, the message sizes and the application and networking configuration.

Because this project was defined before the industry-standard STAC-M2 Benchmark specifications were defined (see www.STACresearch.com/m2), these tests followed 29West's own test procedures with some modifications by STAC. 29West provided the publishing and subscribing applications, which they instrumented with the STAC Observer Library. This library was configured to use a timing methodology involving PCI-based timing cards. This enabled us to measure one-way latency with high accuracy. Other STAC Tools analyzed the output of the STAC Observer Libraries to develop latency and throughput statistics. STAC inspected the 29West source code to ensure that publishers and subscribers followed the agreed methodology.

2.1.2 Test Setup

Depending on the test setup, either two or four IBM System x3650 servers were used. Each server ran SLERT on two Xeon X5355 ("Clovertown") processors. Each server had one Solarflare 10GigE NIC and driver connected to a Cisco Catalyst 4900M 10 Gigabit Ethernet switch. Some tests used the OpenOnload acceleration library from Solarflare.

Figure 2-1 shows the basic server components used in testing. The publisher and subscriber test applications integrated 29West's Latency Buster Middleware (LBM) for sending and receiving messages across the 10GigE network on a specified UDP multicast channel. The applications also integrated the STAC Observer Library, which captured message ID and timestamp information for each message as it was published to or received from the LBM API. The STAC Observer library stored all observations in RAM during the test and then wrote the observations to disk after the test completed. 29West performed the integration work, and STAC inspected the final code.

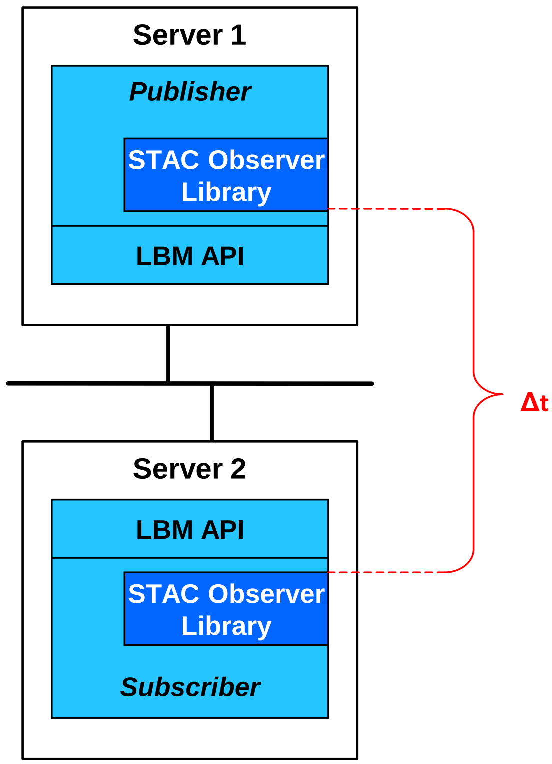


Figure 2-1 – Basic Test Setup

The publisher application provided messages to the LBM API at a steady-state rate using a single subject. Each test ran for 30 seconds. A message payload of the configured size was created in memory. For each message, the application provided LBM with a header and a pointer to this payload. The LBM call was set to non-blocking. The application called the STAC Observer Library immediately before each message was passed to the LBM API send function (i.e., the "send time"). During testing, the publisher application checked the rate at which messages were actually given to the LBM API each second. If this rate was less than the target rate for three consecutive seconds, the publisher would print an error to the console and exit. (This condition did not occur during testing.)

The subscriber application registered with the LBM API to receive messages for the given subject available over a configured UDP multicast channel. For each received message, the subscriber application decoded the message header (but not the payload) and then immediately called the STAC Observer library.

The latency of each message (Δt) was calculated in post-processing as the difference between the subscriber's receive timestamp and the publisher's send timestamp. The average message rate was calculated by the STAC Tools during post-processing and manually checked against the expected rate.

2.1.3 Test Types

The following table summarizes the main variables in the four types of tests:

Pubs: Subs	Variable	Latency-Optimized Configs	Throughput-Optimized Configs
1:1	Rate	Range up to max rate sustainable for 1024-byte payloads	Range up to max rate sustainable for 1024-byte payloads
	Message size	64B, 128B, 256B, 512B, 1024B	64B, 128B, 256B, 512B, 1024B
	Network stack	OpenOnload	OpenOnload
	MTU	1500	9000
3:3	Rate	25K, 50K, 100K, 125K	
	Message size	64B, 128B, 256B, 512B, 1024B	64B, 1024B
	Network stack	OpenOnload	OpenOnload and Kernel
	MTU	1500	9000

Table 2-1 – Test Types

In Table 2-1, and other tables in this document, "1:1" is used as short-hand for a "single publisher, single subscriber" setup and "3:3" is used to denote a "three publisher, three subscriber" setup.

Messages sizes were chosen to reflect a range typical for market data and internal data applications in a trading firm.

Message rates were chosen as follows:

- For the 1:1 and 3:3 latency-optimized configurations, first run tests to find the highest rate that can be sustained with no data loss at 1024B while maintaining max latency levels under 1ms. Then choose three other rates that are lower than this maximum to provide a reasonable spread of rates.
- For the 1:1 throughput-optimized configurations, run tests to find the highest rate that can be sustained with no data loss reported by LBM for each payload size. Then for each payload size, run at each of these rates (those that do not exceed the maximum for that payload size) in order to enable comparisons.

- For the 3:3 throughput-optimized configuration, we found the highest sustainable rate for each publisher/subscriber pair using 64B and 1024B message sizes.

For each configuration, the vendors tuned the stack to minimize latency or maximize throughput (details are in sections 2.2.3, 2.2.6, and 2.2.8):

- The latency-optimized configuration used LBM, OpenOnload and NIC configuration settings that minimized the number of messages sent in a network packet, thus minimizing the time required for a message to be sent by a publisher. Subscribers were configured to receive messages as soon as they were available from the network.

- The throughput-optimized configuration used LBM, OpenOnload and NIC configurations settings that maximized the number of messages that could be sent in each network packet. This included an MTU of 9000. Subscribers were configured to receive messages as soon as they were available, relying on upstream message packing to economize system reads.

Figure 2-2 and 2-3 show the main components of the 1:1 and 3:3 setups.

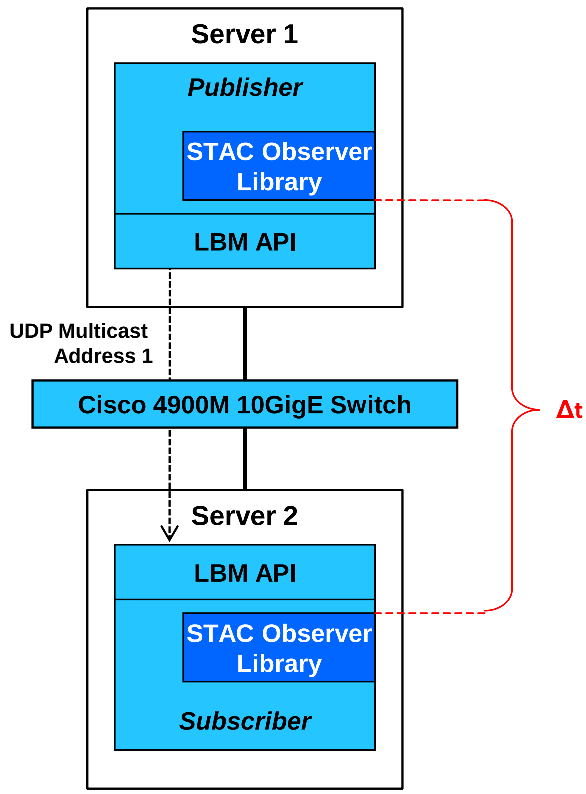


Figure 2-2 – Single Publisher, Single Subscriber (1:1) Setup

In the 1:1 configuration, a single publisher application on a single server sent messages to a single subscriber application on a single server, using one UDP multicast address as configured through LBM.

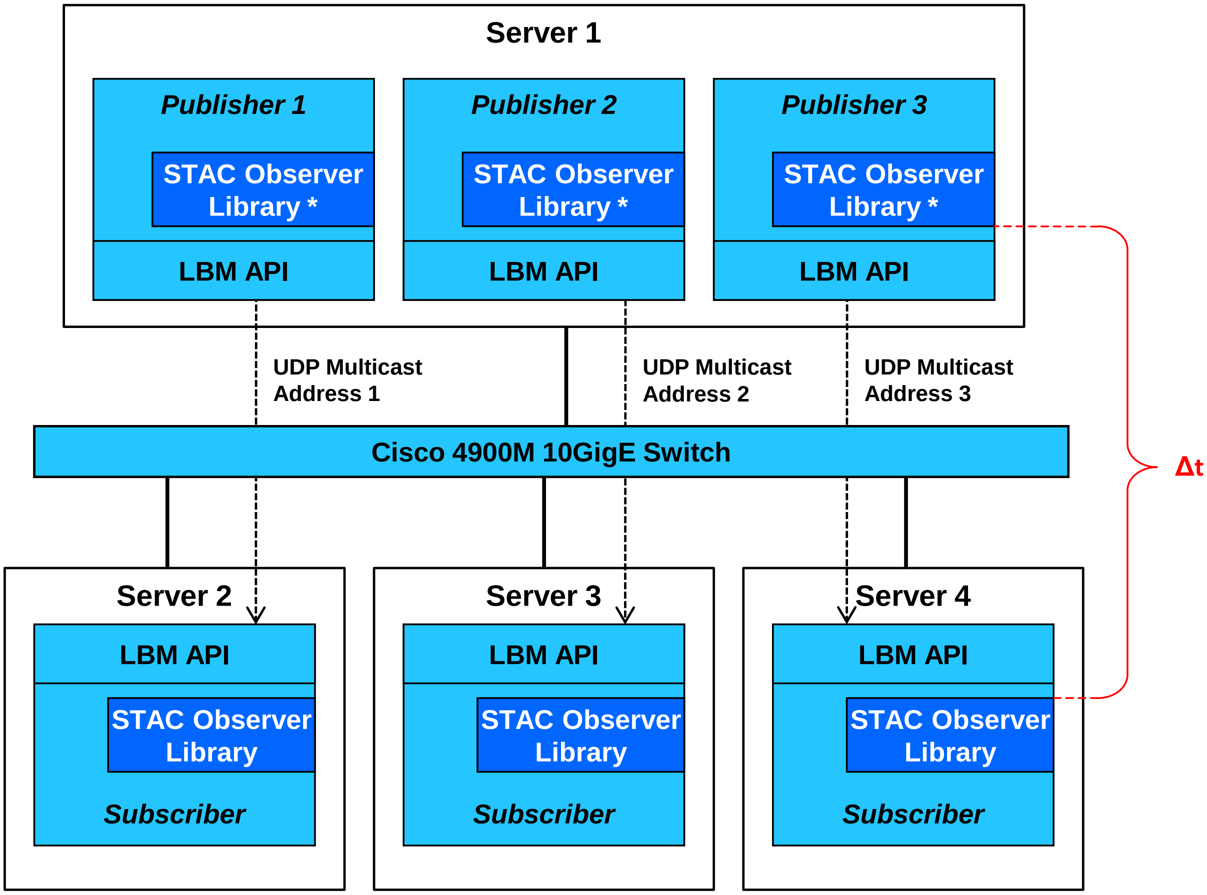


Figure 2-3 – Three Publisher, Three Subscriber (3:3) Setup

In the 3:3 configuration, three publisher application instances on a single server sent messages on unique UDP multicast addresses. Traffic from all three publishers ran through one 10GigE NIC and port. One subscriber application instance on each of three servers received messages from one of the UDP multicast addresses—i.e., there was a 1:1 mapping of publisher and subscriber applications.

Each publisher and subscriber messaging pair is considered a data "path," as follows:

- Data Path 1: publisher 1 on Server 1 to subscriber on Server 2
- Data Path 2: publisher 2 on Server 1 to subscriber on Server 3
- Data Path 3: publisher 3 on Server 1 to subscriber on Server 4

The Multiple Publisher Test Configuration

The 3:3 tests were designed to emulate real-world scenarios in which multiple processes publish from the same box. This is a common deployment for market data feed handlers. In this project, all three publisher processes utilized a single instance of OpenOnload, a single NIC port, and a single port on the Cisco Catalyst 4900M.

The one-to-one mapping of publishers to subscribers in these tests emulates a set of real-world use cases in which subscriber interest and publisher subjects can be neatly mapped to one another via discrete multicast addresses. Examples include applications that are interested in specific indices or instruments on a given exchange.

Other real-world use cases involve subscription to a subset of the subjects on multiple multicast addresses. However, testing these more complicated use cases was beyond the scope of this project.

During a given test run, all subscribers captured observations; however, due to constraints in this version of the STAC Observer Library, only one of the publishers was able to capture observations in a given run. We varied the publisher used for observation, from run to run.

2.1.4 Procedures

A “test suite” comprised the tests defined for the given test type. For example, a test run of the “1:1, Latency Optimized” test type involved running individual tests for all the combinations of message rates and sizes that are defined in Table 2-1 for the test type. We ran each test suite twice and combined the results (e.g., the max latency is the max across both runs).

Before the start of each test suite, all SUT servers were rebooted, and the OS and NIC’s were configured as appropriate for the test type.

For each 1:1 test, the following run procedure was executed:

1. Start the subscriber on Server 2.
2. Start the publisher on Server 1, configured to send messages using a specified size and rate.
3. Publisher sends messages at the given rate. Subscriber receives messages. Publisher and subscriber both capture observations for all messages.
4. Publisher stops sending messages after 30 seconds.
5. Publisher and subscriber write observations to disk and stop.
6. Copy all observation files to an analysis server and analyze the data using STAC Tools.

For each 3:3 test, the following run procedure was executed:

1. Start three subscribers, one each on Servers 2, 3 and 4.
2. Start three publishers on Server 1, each configured to send messages of the same size.
3. Each publisher sends messages at the same given rate. Subscribers receive messages. One publisher and all subscriber applications capture observations.
4. Publishers stop sending messages after 30 seconds.

5. Publishers and subscribers write observations to disk and stop.
6. All observation files are copied to an analysis box and STAC analysis tools are run on the data.

2.1.5 Time synchronization

The SUT was synchronized using a hardware-based methodology that relies on time-synchronization cards in each server fed by a high-precision master clock. The STAC Observer Library interfaces to the timing cards and records observations with a high precision timestamp. Using an external measurement device, we determined that the synchronization error was ± 1 microsecond at each instance of the STAC Observer Library, or ± 2 microseconds for latencies measured by two library instances. This was corroborated by running several tests in each direction (switching the publishing and subscribing roles of the servers) and comparing the latency minimums, which did not differ by more than 2 microseconds.

2.1.6 Limitations

Like most lab projects, the tests in this project differed from the ideal procedure in a number of respects, usually due to time or resource constraints:

- **Vendor-supplied publishers and receivers.** Use of 29West's code for publishers and consumers was a minor limitation, since STAC code did not control the message supply code. However, STAC did inspect the source code of these applications to ensure proper use of the STAC Observer Library.

- **Single subject per publisher.** Each publisher sent, and each subscriber received messages on just one topic. In the real world, publishers and subscribers typically use hundreds to hundreds of thousands of subjects. Studies of other messaging systems have shown that the number of subjects can have a substantial impact on performance. 29West claims this is true to a much lesser extent with LBM than with other systems.

- **Multicast subscriptions.** Each subscriber in the 3:3 tests consumed all messages on one multicast address, which corresponded to a single publishing application. As noted in Section 2.1.3, this simulates a certain set of real-world use cases but does not reflect the more complicated, and perhaps more common, use cases in which subscribers subscribe to a subset of subjects from multiple multicast channels, supplied by multiple publishers.

- **Steady-state rate.** Real market data environments are characterized by considerable variation in the rate of incoming messages. By contrast, the publishers in these tests were configured for steady-state supply of messages.

- **Performance impact of STAC Observer Library.** The STAC Observer Library performs work related to time-stamping messages. This function takes some amount of CPU time and reduces the amount of CPU that is available to the application. However, many real trading applications also timestamp each outgoing or incoming message, so this load is not unrealistic. The STAC Observer Library has been carefully designed to limit the extent to which the tool itself impacts the underlying performance of the system. In these tests, it retained all observations in memory, thus eliminating the need for disk writes. We have determined empirically that each call to the STAC Observer Library takes between 150-200 nanoseconds. Although the processing time is minimal, for a system that is operating at maximum CPU, this overhead may reduce the achievable performance.

- **Test-run length.** Test runs were 30 seconds long. Longer test runs can sometimes reveal additional insight into system performance.

2.2 System Specifications

2.2.1 Servers

Each of the servers in the test harness had the following specifications:

Vendor Model	IBM System x3650 Server
Processors	2
Processor Type	Quad-Core Intel Xeon X5355 2.66 GHz (codename "Clovertown")
Cache	32KB L1 Cache per core 8MB L2 Cache per processor with 4 MB per 2 cores
Bus Speed	1.333 GHz
Memory	8 GB RAM, 266 MHz
Eth0	Broadcom Corporation NetXtreme II BCM5708 Gigabit Ethernet
Eth2	Solarflare Communications SFE 4002
NIC Note	For all tests, management traffic was directed at eth0 and SUT traffic was directed at eth2
BIOS	BIOS Information Vendor: IBM Version: GGE142AUS-1.13 Release Date: 12/03/2008 Firmware version: 1.33
Rack Units	2

2.2.2 Networking

10GigE Switch	Cisco Catalyst 4900M 10 Gigabit Ethernet Switch IOS: cat4500e-entservicesk9-mz.122-46.SG.bin All ports used on VLAN 100 MTU set to 9198 Default 16MB buffer allocated to single queue
10GigE NIC	Solarflare Communications Solarstorm SFE4002
10GigE NIC driver	sfc, version: 2.3.19.1053
10GigE NIC firmware	n/a
10GigE NIC note	All interfaces were connected to the switch via 10GBASE-SR interfaces
1GigE NIC	Broadcom Corporation NetXtreme II BCM5708
1GigE NIC driver	bnx2, version 1.6.7
1GigE NIC firmware	1.9.6

2.2.3 Networking Interface Configurations

Any settings changed from the defaults are noted below

(eth2) ring buffer size	For runs not using OpenOnload: 1k RX ring size For runs using OpenOnload: See "EF_RXQ_SIZE" setting in section 2.2.6
(eth2) MTU	Latency-optimized config: MTU=1500 Throughput-optimized config: MTU=9000
(eth2) OpenOnload	OpenOnload v2.3.19.1053 20090303 - The application command line was prefixed with "onload", which sets up the application to use OpenOnload rather than the network protocol stack.

2.2.4 Operating System

Version	SUSE Linux Enterprise Real Time 10 SP2 - 64-bit Kernel - 2.6.22.19-20090112_SLERT10_SP2_BRANCH-rt
General OS Services	For all tests, a script was run to stop unnecessary OS services. The stopped services were: acpid alsasound autofs bluetooth conman cpuspeed cron cups cupsrenice dhcdbd dnsmasq dund firstboot hidd ip6tables ipmi irda irqbalance irq_balancer kudzu libvirtd lvm2-monitor mcstrans mdmonitor mdmpd messagebus multipathd netconsole netfs netplugd nfs nfslock nscd oddjobd pand pcscd portmap postfix powersaved psacct rdisc readahead_early readahead_later restorecond rhnsd rpcgssd rpcidmapd rpcsvcgssd saslauthd sendmail slpd smbfs suseRegister wpa_supplicant xfs ypbind yum-updatesd novell-zmd
System Management Interrupts	"smictrl -s 0" was run to disable SMI's
Shielding	For all tests, cores 0 – 6 were shielded using the commands: "cset shield –r; cset shield -c 0-6 -k on"
/proc/interrupts and realtime IRQ threads	The following procedure was used to bind interrupt processing to specific cores: 1. Set the smp_affinity of all irq's found in /proc/irq to core 7 2. Bind all OS realtime IRQ-<irq> threads to core 7 3. Set smp_affinity of SolarFlare irq's found in /proc/irq to core 0. In this setup, there were two irqs associated with the Solarflare driver. 4. Bind OS realtime IRQ-<irq> threads associated with Solarflare irq's to core 0

2.2.5 TCP and UDP Buffers – key parameters

No system-wide modifications were made.

2.2.6 OpenOnload Configuration Parameters

	Test Setup	Test App Type	OpenOnload configuration
Latency-optimized	1:1 and 3:3	Publisher	EF_INT_DRIVEN=1
		Subscriber	EF_POLL_USEC=100
			EF_INT_REPRIME=0
			EF_RXQ_SIZE=2048
Throughput-optimized	1:1	Publisher	EF_INT_DRIVEN=1
		Subscriber	EF_POLL_USEC=100
			EF_INT_REPRIME=0
			EF_RXQ_SIZE=2048
	3:3 - 1024B	Publisher	EF_INT_DRIVEN=1
			EF_RXQ_SIZE=4096
		Subscriber	EF_POLL_USEC=100
			EF_INT_REPRIME=0
			EF_RXQ_SIZE=4096
	3:3 – 64B	Publisher	EF_INT_DRIVEN=1
			EF_INT_REPRIME=0
			EF_POLL_USEC=0
			EF_POLL_ON_DEMAND=0
			EF_RXQ_SIZE=4096
		Subscriber	EF_INT_DRIVEN=1
			EF_INT_REPRIME=0
			EF_POLL_USEC=0
			EF_POLL_ON_DEMAND=0
			EF_RXQ_SIZE=4096

2.2.7 29West/LBM Software

LBM Test Tools	staclbmpub and staclbmsub
LBM API	Version 3.3.9

2.2.8 29West/LBM Configuration Parameters

	Test Setup	Test App	LBM configuration
Latency-optimized	1:1 3:3	Publisher	source implicit_batching_minimum_length 1
			context transport_lbtrm_receiver_socket_buffer 8388608
			context transport_session_multiple_sending_threads 0
			context transport_datagram_max_size 1472
			context fd_management_type poll
			source transport_lbtrm_ignore_interval 15
		Subscriber	context transport_lbtrm_receiver_socket_buffer 8388608
			context fd_management_type poll
Throughput-optimized	1:1 3:3	Publisher	source implicit_batching_minimum_length 8192
			context transport_lbtrm_receiver_socket_buffer 8388608
			context transport_session_multiple_sending_threads 0
			context fd_management_type poll
	1:1 3:3 - 1024B 3:3 - 64B/Kernel	Subscriber	context transport_lbtrm_receiver_socket_buffer 8388608
			context fd_management_type poll
	3:3 - 64B/Onload	Subscriber	context transport_lbtrm_receiver_socket_buffer 12388608
			context fd_management_type poll
			receiver delivery_control_maximum_burst_loss 2048

2.2.9 Publisher and Subscriber Application CPU Bindings

STAC Observer threads	*For all tests, publisher and subscriber threads associated with the STAC Observer library:* taskset to core 6
Publisher 1 threads	*For latency-optimized config:* taskset to core 1 *For throughput-optimized config:* tasket to core 0 & 1 *For throughput-optimized,3:3, OpenOnload, 64B config:* tasket to core 1
Publisher 2, 3	*For all tests:* taskset to core 2
Subscribers 1, 2 and 3	*For all tests:* taskset to core 1

3. Results

3.1 Latency-optimized configurations

"Send-to-end" latency is defined as the delta between the time an update is posted by the publisher application to its API and the time the same update is received by the consuming application from its API. In each of these latency tests, LBM, OpenOnload, and SLERT were configured to deliver the lowest latency at the expense of throughput.

3.1.1 Max throughput per path

As discussed in Section 2.1.3, our first task was to determine the maximum throughput that this configuration could sustain without maximum latency exceeding one millisecond while carrying 1024-byte payloads. The result was 125,000 messages per second. We therefore used this as the maximum rate tested across the various payload sizes for the latency-optimized configuration.

3.1.2 Single Publisher/Single Subscriber (1:1)

Table 3-1 records the latency statistics combined from two test runs of the single publisher/single

Payload Size (bytes)	Target Rate (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
64	25,000	13	14	4	11	68	30	46
	50,000	13	15	5	11	107	36	55
	100,000	12	15	5	10	738	39	49
	125,000	13	15	5	11	635	38	58
128	25,000	13	15	5	11	79	39	57
	50,000	13	15	5	11	96	39	58
	100,000	13	15	5	11	673	37	61
	125,000	13	15	5	10	653	36	60
256	25,000	14	16	6	11	112	41	59
	50,000	13	15	6	11	666	42	52
	100,000	14	16	5	11	592	37	54
	125,000	13	16	5	11	655	39	58
512	25,000	15	17	6	12	87	35	52
	50,000	14	17	6	12	131	39	63
	100,000	14	16	5	12	617	39	62
	125,000	14	16	5	12	676	38	60
1024	25,000	16	19	6	14	123	46	59
	50,000	16	18	6	13	680	44	61
	100,000	16	18	5	14	633	40	59
	125,000	16	18	6	14	712	42	61

Table 3-1: LBM latency in 1:1 latency optimized configuration on 10GigE Solarflare

subscriber tests as described in Section 2.1. Mean latency ranged from 14 to 19 microseconds and median latency ranged from 12 to 16 microseconds. In summary, the system demonstrated very consistent latency at the four message rates for the same message size.

Figure 3-1 and Figure 3-2 present, respectively, the mean latency and 99.9th percentile latency observed for each payload size and rate. We observed that for each payload size, as message rates increased, the latency remained almost constant. Figure 3-3 plots mean latency against message size for the message rates reported. Mean latency increases nearly linearly with message size, but only by 4 microseconds as the size increases by a factor of 16. All three charts indicate that for this range of message rates, the latency of this SUT configuration is essentially independent of message rate.

Figure 3-4 plots the standard deviation of the system against message rate for each payload size. It shows that there is typically between 5 and 6 microseconds of standard deviation (or "jitter") at these rates. It also shows that jitter is relatively constant irrespective of message rate, except for 64-byte payloads, which enjoy slightly lower jitter at rates below 100Kmps.

Table 3-1 shows that the maximum latency generally increases with rate, while the mean, median, and percentiles remain very flat. Figures 3-5 and 3-6 plot histograms of the high-rate cases: 64-byte messages at 125Kmps and 1024-byte messages at 125Kmps. The histograms show two very tight peaks in the low end of the distribution. Inspection of the histogram data revealed that the maxes were extreme outliers. Of the 7.5 million data points obtained from two runs of the 1024B/125Kmps test, the highest latency was 121 microseconds, except for 28 outliers with latencies from 553 to 712 usec. The maxes in the 64B/125K histogram data were a similar small clutch of outliers.

The source of the outliers was not readily apparent. We verified that there were no NAKs and retransmissions during the runs by using 29West utilities. Additionally, we loaded the SLERT user space module for turning off System Management Interrupts (SMI). We were unable to attribute these occurrences to a specific component in the SUT.

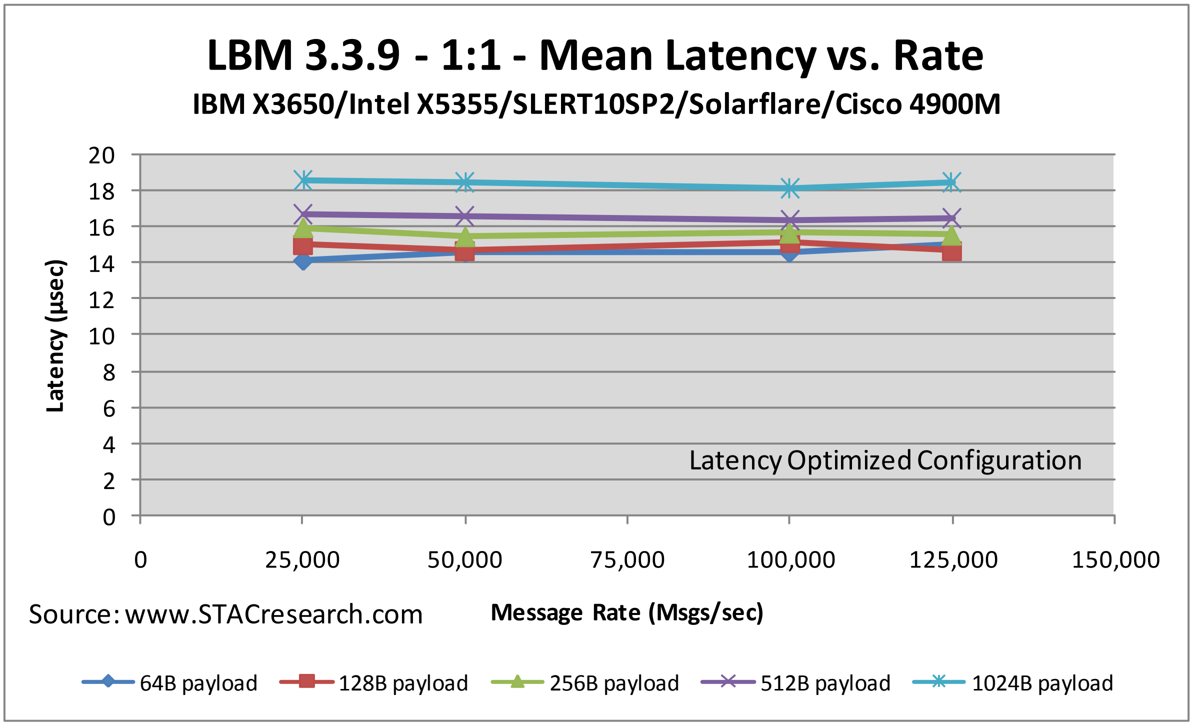


Figure 3-1

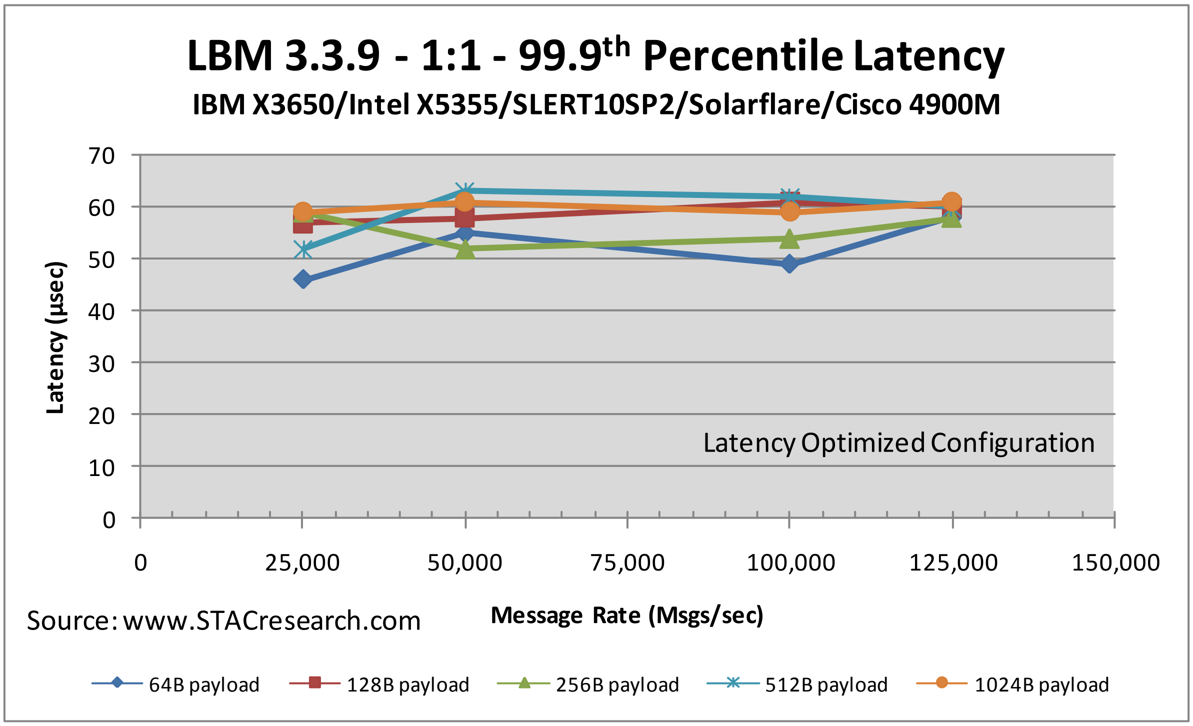


Figure 3-2

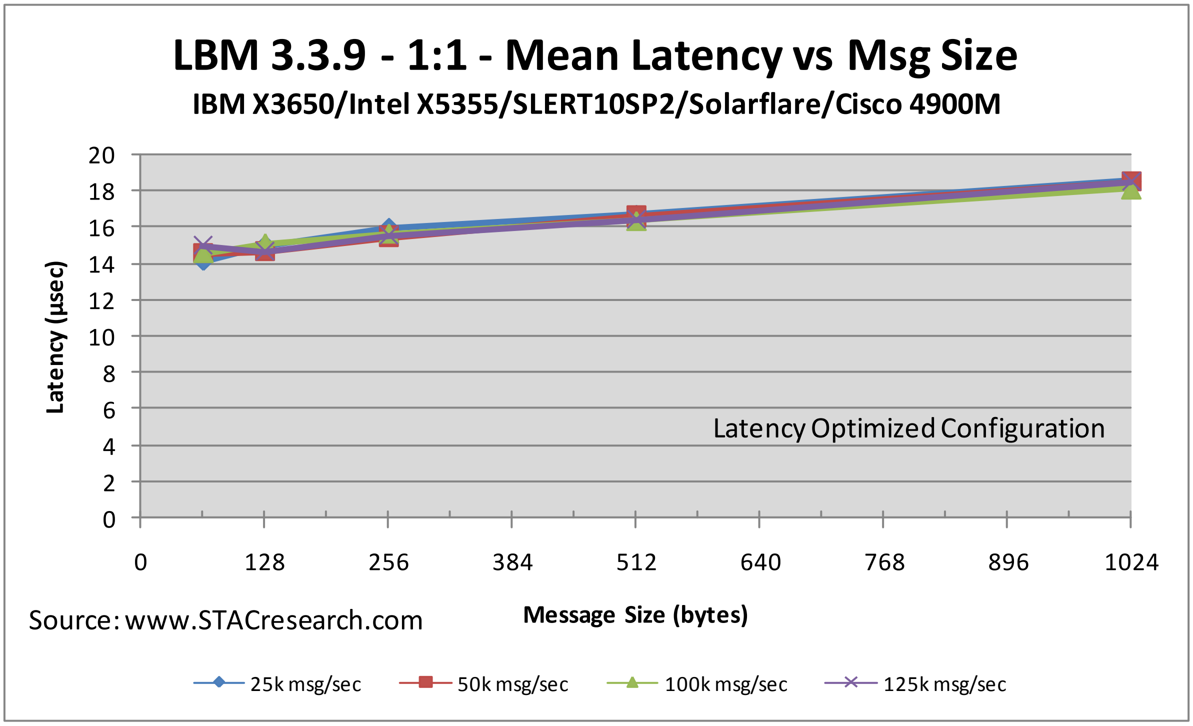


Figure 3-3

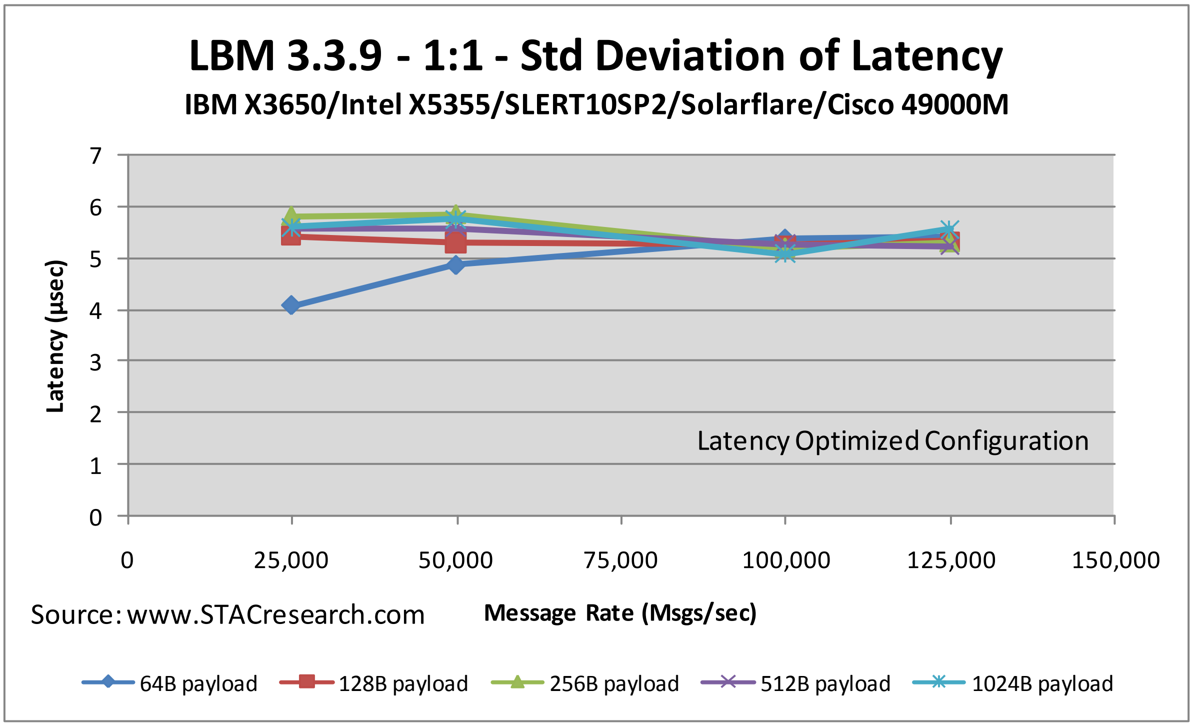


Figure 3-4

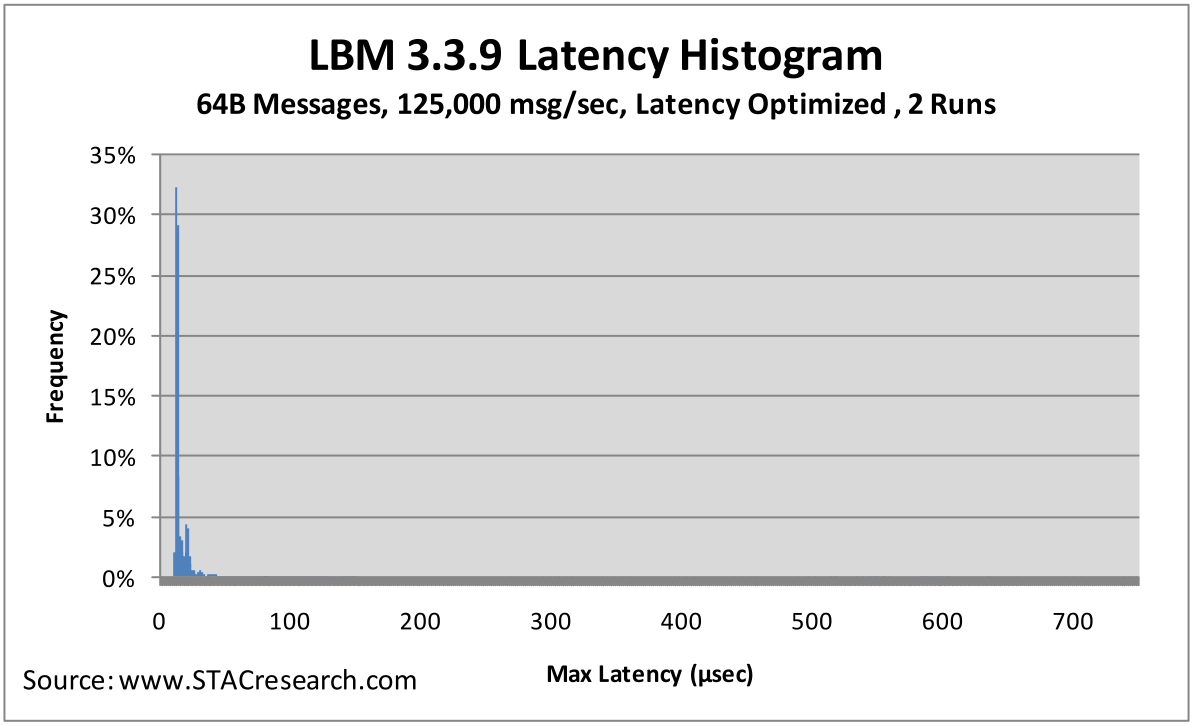


Figure 3-5

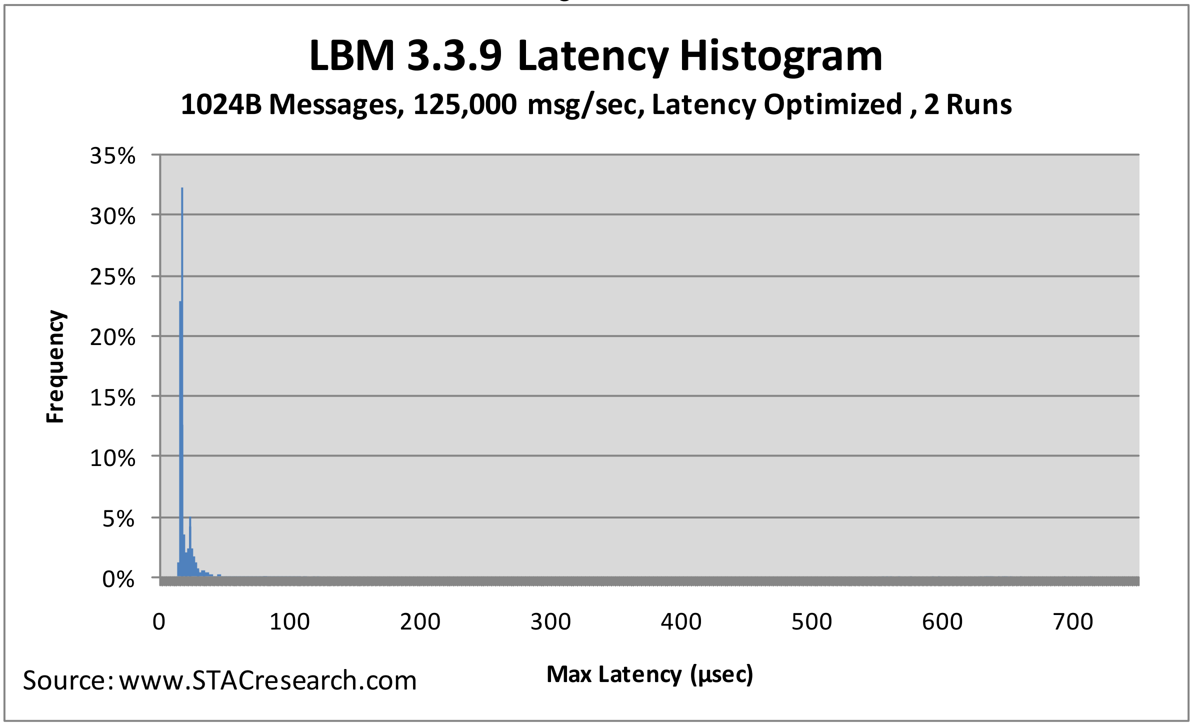


Figure 3-6

3.1.3 Three Publishers, Three Subscribers

In these tests, described in Section 2.1.3, three publishers sent messages from the same server to 3 subscribers, each on its own server. Each subscriber received data from a single publisher. Table 3-2 shows the observations of the combined runs from path 1. We verified that path 2 & path 3 had similar results. Full results from all paths are presented in the Appendix.

Payload Size (bytes)	Target Rate (msg/sec)	*Median* (μsec)	*Mean* (μsec)	*StdDev* (μsec)	*Min* (μsec)	*Max* (μsec)	*99%* (μsec)	*99.9%* (μsec)
64	25000	13	14	4	11	591	29	50
	50000	12	14	5	11	75	36	55
	100000	13	14	5	10	669	37	57
	125000	13	15	5	10	609	38	60
128	25000	13	15	6	11	557	42	60
	50000	13	15	6	10	586	40	62
	100000	13	15	6	11	774	41	69
	125000	13	15	5	10	677	38	61
256	25000	13	15	4	11	94	32	52
	50000	13	15	5	11	632	42	62
	100000	13	15	5	11	770	36	59
	125000	14	16	5	11	766	39	61
512	25000	14	16	4	12	93	33	53
	50000	14	16	5	12	602	34	63
	100000	15	17	6	12	752	43	77
	125000	15	16	5	12	795	39	61
1024	25000	16	18	6	14	647	39	56
	50000	16	18	5	13	102	37	53
	100000	17	19	6	14	843	43	66
	125000	17	19	5	14	917	41	62

Table 3-2: LBM latency in 3:3 latency optimized configuration on 10GigE Solarflare

To compare latencies of the 3:3 case to those of the 1:1 case, Table 3-3 tabulates the differences in results for the same tests. The table shows that the mean, median, and standard deviation are indistinguishable from the Single Publisher case. The maxes in the 3:3 case are generally higher, which probably reflects occasional resource contention in the publisher. However, the 99^{th} percentile is not generally higher, and the 99.9^{th} percentiles are only higher by a few microseconds. However, the 99th percentile is not generally higher, and the 99.9th percentiles are only higher by a few microseconds, which suggests that the maxes are once again outliers.

Payload Size (bytes)	Target Rate (msg/sec)	*Δ Median* (µsec)	*Δ Mean* (µsec)	*Δ StdDev* (µsec)	*Δ Min* (µsec)	*Δ Max* (µsec)	*Δ 99%* (µsec)	*Δ 99.9%* (µsec)
64	25000	0	0	0	0	523	-1	4
	50000	-1	0	0	0	-32	0	0
	100000	1	0	0	0	-69	-2	8
	125000	0	0	0	-1	-26	0	2
128	25000	0	0	0	0	478	3	3
	50000	0	0	0	-1	490	1	4
	100000	0	0	0	0	101	4	8
	125000	0	0	0	0	24	2	1
256	25000	-1	-1	-2	0	-18	-9	-7
	50000	0	0	0	0	-34	0	10
	100000	-1	0	0	0	178	-1	5
	125000	1	0	0	0	111	0	3
512	25000	-1	-1	-1	0	6	-2	1
	50000	0	-1	0	0	471	-5	0
	100000	1	1	1	0	135	4	15
	125000	1	0	0	0	119	1	1
1024	25000	0	0	0	0	524	-7	-3
	50000	0	-1	-1	0	-578	-7	-8
	100000	1	0	0	0	210	3	7
	125000	1	0	0	0	205	-1	1

Table 3-3: Latency results for 3:3 configuration minus results for 1:1 configuration

3.2 Throughput-optimized configurations

Throughput is defined as the number of messages published from a server and received without message loss. In each of these throughput tests, LBM, OpenOnload, and SLERT were configured to optimize throughput at the expense of latency.

3.2.1 Max Throughput

As discussed in Section 2.1.3, our first task was to determine the maximum throughput that this configuration could sustain at each payload size without reporting data loss. The results are shown below in Table 3-4 and Figure 3-9. We therefore used these rates in the throughput-optimized configuration.

Payload Size	Max Rate (msg/sec)
64 byte	860,000
128 byte	790,000
256 byte	650,000
512 byte	340,000
1024 byte	270,000

Table 3-4: Maximum throughput rates per payload size

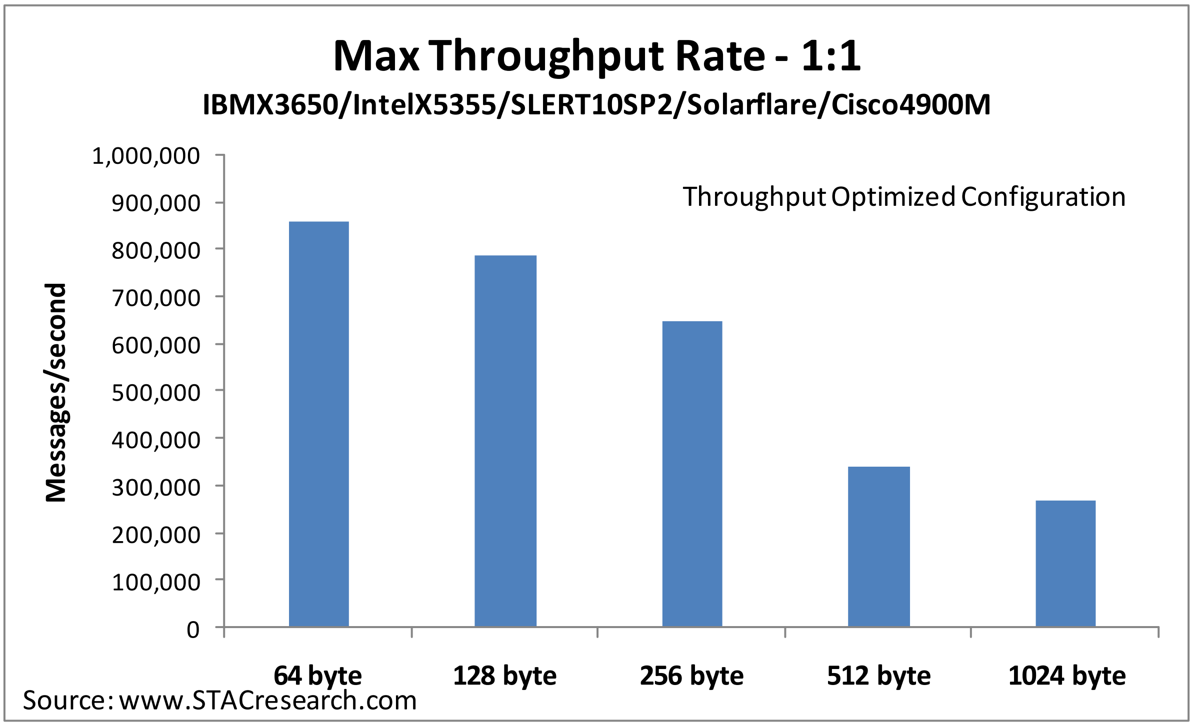


Figure 3-9

3.2.2 Single Publisher/Single Subscriber (1:1)

Table 3-5 records the latency statistics for the single publisher, single subscriber throughput tests as described in Section 2.1. The rates chosen were the maximum rates for each of the payload sizes as shown above. Only results from rates that could be sustained without data loss are reported in table 3-4.

Figure 3-10 shows the mean latency curve for each payload size. The max message rate achieved was 860Kmps. At this rate, the SAR data showed that the publisher was CPU bound. This suggests that on newer CPUs, higher rates might be achievable.

Table 3-5 shows, as expected, that latency increased with message rate, with mean latency ranging from 13 to 136 microseconds.

As expected, maximum latencies of this throughput-optimized configuration were significantly higher than those of the latency-optimized configuration. The 270Kmps rate and the 340Kmps rate consistently showed increases in the max, 99.9th percentile, and 99th percentile values. We see that the standard deviation also increases at these rates. 29West believes that this may be due to some "resonance" likely from the batching algorithms in the hardware, OS, and, LBM. This will require further investigation.

Payload Size (bytes)	Target Rate (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
64	125,000	13	15	6	11	1005	41	61
	270,000	17	19	8	10	961	44	78
	340,000	22	87	300	11	9678	1340	4025
	650,000	41	60	61	16	1178	330	605
	790,000	62	74	36	23	524	189	270
	860,000	134	136	15	55	379	177	199
128	125,000	13	16	8	11	11129	40	61
	270,000	19	36	162	11	8398	150	2989
	340,000	24	52	133	11	3477	488	1903
	650,000	62	69	33	17	607	174	257
	790,000	92	95	11	28	362	134	155
256	125,000	13	16	6	11	571	39	59
	270,000	23	36	74	11	4948	185	1324
	340,000	30	41	50	12	1402	236	726
	650,000	68	72	11	26	337	110	132
512	125,000	15	17	6	12	204	41	60
	270,000	31	34	14	12	297	79	117
	340,000	51	50	16	15	299	104	135
1024	125,000	16	20	7	14	273	45	62
	270,000	44	51	16	19	259	113	140

Table 3-5: LBM latency in 1:1 throughput optimized configuration on 10GigE Solarflare

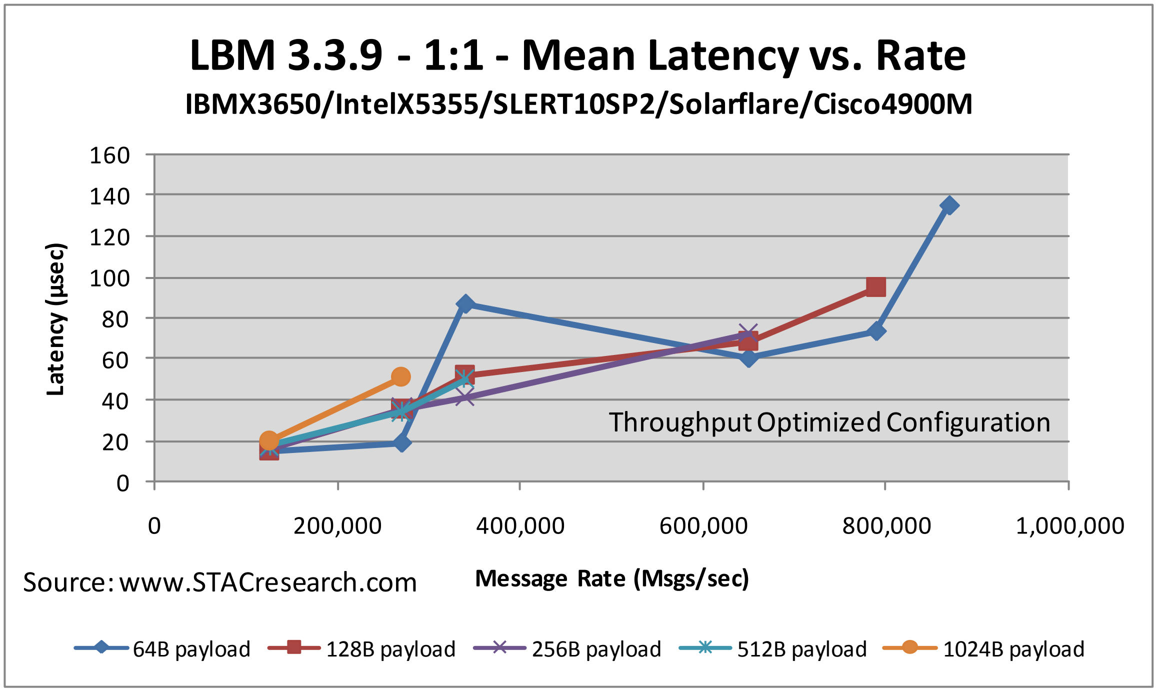


Figure 3-10

3.2.3 Three Publisher/Three Subscriber (3:3), 64-Byte Messages

Our goal with this test was to maximize throughput of 64-byte messages from a single server using multiple publishers. As table 3-5 shows, the maximum message rates are achieved with the smallest messages. We ran configurations using OpenOnload as well as the standard kernel IP stack and three publishers in each configuration. Table 3-6 shows the resulting rates by publisher and in aggregate. With OpenOnload, we achieved 1.86 million mps, which corresponded to 0.95 Gigabits per second (Gbps). With the standard kernel stack, we were able to publish 2.46 million mps (1.26 Gigabit per second).

The OpenOnload configuration in this 3:3 test differed slightly from that of the 1:1 tests discussed in section 3.2.1. Although 1.86 million mps was the highest aggregate throughput that could be achieved with OpenOnload, the maximum rate of 620Kmps per publisher was lower than the maximum rate achieved in the previous 1:1 throughput tests at this message size. Unfortunately, further time to optimize was not available in the project and this may be revisited in a future project.

As discussed in Section 2.1.3, one path per test run was instrumented to measure latency. However, when paths 2 and 3 were instrumented, we found that the max rates for these paths were lower than when they were not instrumented. This is not surprising, given that the STAC Observer Library does impose some additional load, as discussed above. Path 1 results are presented in Table 3-7 for OpenOnload and Table 3-8 for the Linux Kernel Stack. The full results are presented in the Appendix. Although latency observations for both OpenOnload and the kernel stack are presented, no meaningful comparison can be made because they are measured at different rates.

	Publisher Message Rate (mps)			Aggregate Message Rate (mps)	Aggregate Bandwidth (Gbps)
	Publisher 1	Publisher 2	Publisher 3		
OpenOnload, 64B messages	620,000	620,000	620,000	1,860,000	0.95
Kernel, 64B messages	800,000	830,000	830,000	2,460,000	1.26

Table 3-6: Throughput test results for 3:3 test with 64-byte messages

Path	Actual Rate	Median	Mean	StdDev	Min	Max	99%	99.9%
1	619,894	103	106	22	37	750	183	235
2	559,905	98	3,042	25,667	39	317,805	161,608	297,906
3	559,905	97	99	19	39	831	156	220

Table 3-7: Latency for 3:3 throughput-optimized config using OpenOnload with 64-byte messages

Path	Actual Rate	Median	Mean	StdDev	Min	Max	99%	99.9%
1	799,857	181	179	30	68	436	241	264
2	794,222	195	197	28	69	920	265	292
3	797,507	180	185	27	55	892	257	286

Table 3-8: Latency for 3:3 throughput-optimized config using kernel stack with 64-byte messages

3.2.4 Three Publisher/Three Subscriber (3:3), 1024-Byte Messages

This test had the same goal as the previous test (maximizing throughput from a single server using three publishers) but with 1024-byte messages. Again, we tested configurations using both OpenOnload as well as the standard kernel IP stack.

Table 3-9 shows the resulting rates per publisher and in aggregate. With OpenOnload, we achieved 550,000 mps (4.51 Gbps). With the standard Linux kernel stack, we achieved 520,000 mps (4.26 Gbps).

The OpenOnload configuration in this 3:3 test was the same as in the 1:1 throughput tests discussed in section 3.2.1. Publisher 1 achieved nearly the same rate as in the previous tests. However publishers 2 & 3 were unable to reach the same rate. With OpenOnload, we found that the highest rate could be achieved when the publisher was on a sibling core to the interrupt for OpenOnload. Unlike the 64-Byte tests, instrumenting Paths 2 and 3 did not decrease their maximum throughput. Path 1 results are presented here in Table 3-10 for OpenOnload and Table 3-11 for the Linux Kernel Stack. The full results are presented in the Appendix.

Although latency observations for both OpenOnload and the kernel stack are presented, no meaningful comparison can be made because they measured are at different rates.

	Publisher Message Rate (mps)			**Aggregate message rate (mps)**	**Aggregate Bandwidth (Gbps)**
	Publisher 1	Publisher 2	Publisher 3		
OpenOnload, 1024B messages	250,000	150,000	150,000	550,000	4.51
Kernel, 1024B messages	180,000	170,000	170,000	520,000	4.26

Table 3-9: Throughput test results for 3:3 test with 1024-byte messages

Path	Actual Rate	Median	Mean	StdDev	Min	Max	99%	99.9%
1	249,958	47	54	17	17	266	115	148
2	149,975	51	59	22	20	1063	136	209
3	149,975	51	59	20	21	983	130	190

Table 3-10: Latency for 3:3 throughput test using OpenOnload with 1024-byte messages

Path	Actual Rate	Median	Mean	StdDev	Min	Max	99%	99.9%
1	179,970	93	94	16	37	285	139	163
2	167,997	108	112	25	50	989	191	225
3	168,514	108	113	25	50	954	193	228

Table 3-11: Latency for 3:3 throughput test using kernel stack with 1024-byte messages

4. Appendix A – Detailed Test Results

4.1 1:1, Latency-Optimized Config

Target Rate (msg/sec)	Payload Size (bytes)	Run	*Actual Rate* (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
25k	*64*	*1*	24996	13	14	5	11	61	31	47
		2	24996	13	14	3	11	68	29	36
	128	*1*	24996	13	15	6	11	71	39	57
		2	24996	13	15	5	11	79	31	54
	256	*1*	24996	13	15	5	11	66	32	52
		2	24996	14	17	7	12	112	42	60
	512	*1*	24996	14	17	6	12	63	35	53
		2	24996	15	17	6	13	87	43	52
	1024	*1*	24996	17	19	7	14	71	48	60
		2	24996	16	18	4	14	123	35	49
50k	*64*	*1*	49992	13	15	5	11	78	36	42
		2	49992	13	15	5	11	107	37	55
	128	*1*	49992	13	14	5	11	75	36	58
		2	49992	13	15	5	11	96	39	58
	256	*1*	49992	13	15	5	11	73	34	52
		2	49992	13	16	6	11	666	43	52
	512	*1*	49992	15	16	5	12	91	39	55
		2	49992	14	17	6	12	131	40	63
	1024	*1*	49992	16	18	5	14	646	35	51
		2	49992	16	18	6	13	680	45	62
100k	*64*	*1*	99983	13	15	5	11	578	41	49
		2	99983	12	14	5	10	738	38	50
	128	*1*	99983	13	15	5	11	78	35	61
		2	99983	13	15	5	11	673	38	49
	256	*1*	99983	14	16	5	12	592	42	50
		2	99983	14	16	5	11	116	36	55
	512	*1*	99983	15	17	5	12	617	40	62
		2	99983	14	16	5	12	609	39	54
	1024	*1*	99983	16	18	5	14	633	40	49
		2	99983	16	18	5	14	571	41	60
125k	*64*	*1*	124979	13	15	5	11	635	38	58
		2	124979	13	15	5	11	546	41	60
	128	*1*	124979	12	14	5	10	626	36	60
		2	124979	13	15	5	11	653	37	60

	256	***1***	124979	13	15	5	11	608	40	58
		2	124979	14	16	5	11	655	37	60
	512	***1***	124979	15	17	5	12	676	39	60
		2	124979	14	16	5	12	654	38	60
	1024	***1***	124979	16	19	6	14	712	44	61
		2	124979	16	18	5	14	652	39	61

4.2 3:3, Latency-Optimized Config

4.2.1 Path 1

Target Rate (msg/sec)	**Payload Size (bytes)**	***Actual Rate*** **(msg/sec)**	***Median*** **(µsec)**	***Mean*** **(µsec)**	***StdDev*** **(µsec)**	***Min*** **(µsec)**	***Max*** **(µsec)**	***99%*** **(µsec)**	***99.9%*** **(µsec)**
25k	***64***	24996	13	14	4	11	591	29	50
	128	24996	13	15	6	11	557	42	60
	256	24996	13	15	4	11	94	32	52
	512	24996	14	16	4	12	93	33	53
	1024	24996	16	18	6	14	647	39	56
50k	***64***	49992	12	14	5	11	75	36	55
	128	49992	13	15	6	10	586	40	62
	256	49992	13	15	5	11	632	42	62
	512	49992	14	16	5	12	602	34	63
	1024	49992	16	18	5	13	102	37	53
100k	***64***	99983	13	14	5	10	669	37	57
	128	99983	13	15	6	11	774	41	69
	256	99983	13	15	5	11	770	36	59
	512	99983	15	17	6	12	752	43	77
	1024	99983	17	19	6	14	843	43	66
125k	***64***	124979	13	15	5	10	609	38	60
	128	124979	13	15	5	10	677	38	61
	256	124979	14	16	5	11	766	39	61
	512	124979	15	16	5	12	795	39	61
	1024	124979	17	19	5	14	917	41	62

4.2.2 Path 2

Target Rate (msg/sec)	Payload Size (bytes)	*Actual Rate* (msg/sec)	*Median* (μsec)	*Mean* (μsec)	*StdDev* (μsec)	*Min* (μsec)	*Max* (μsec)	*99%* (μsec)	*99.9%* (μsec)
25k	***64***	24996	13	16	8	11	950	49	62
	128	24996	13	16	7	11	113	47	62
	256	24996	14	17	7	11	101	47	58
	512	24996	15	18	8	12	609	53	63
	1024	24996	17	19	7	14	169	50	61
50k	***64***	49992	13	16	7	11	604	47	58
	128	49992	13	16	7	11	646	48	61
	256	49992	14	17	7	11	595	50	67
	512	49992	15	18	8	12	653	52	64
	1024	49992	17	20	8	14	691	53	68
100k	***64***	99983	13	16	8	11	932	50	66
	128	99983	13	16	8	11	676	52	69
	256	99983	14	17	8	11	737	51	64
	512	99983	15	18	8	12	783	53	65
	1024	99983	17	21	8	14	808	56	72
125k	***64***	124979	13	16	8	11	665	51	64
	128	124979	14	17	8	11	745	52	64
	256	124979	14	18	8	11	731	53	65
	512	124979	15	19	9	12	746	56	75
	1024	124979	17	21	8	14	942	56	66

4.2.3 Path 3

Target Rate (msg/sec)	Payload Size (bytes)	*Actual Rate* (msg/sec)	*Median* (μsec)	*Mean* (μsec)	*StdDev* (μsec)	*Min* (μsec)	*Max* (μsec)	*99%* (μsec)	*99.9%* (μsec)
25k	***64***	24996	13	16	7	11	99	47	67
	128	24996	14	16	6	11	563	45	58
	256	24996	14	17	7	12	107	50	61
	512	24996	16	18	7	13	648	51	64
	1024	24996	17	21	8	14	111	54	69
50k	***64***	49992	13	16	7	11	104	50	64
	128	49992	13	16	7	11	114	50	63
	256	49992	14	18	8	12	124	51	70

	512	49992	15	19	8	13	717	52	67
	1024	49992	17	20	8	14	692	55	68
100k	***64***	99983	13	16	7	11	751	50	63
	128	99983	14	17	8	11	685	52	66
	256	99983	14	17	8	12	738	52	68
	512	99983	15	19	8	13	735	53	68
	1024	99983	17	21	8	14	782	54	63
125k	***64***	124979	13	17	8	11	1009	52	66
	128	124979	14	17	8	11	682	53	66
	256	124979	14	18	8	11	714	54	68
	512	124979	15	19	8	12	827	53	65
	1024	124979	17	21	8	14	821	56	67

4.3 1:1, Throughput-Optimized Config

Target Rate (msg/sec)	**Payload Size** (bytes)	**Run**	***Actual Rate*** (msg/sec)	***Median*** (µsec)	***Mean*** (µsec)	***StdDev*** (µsec)	***Min*** (µsec)	***Max*** (µsec)	***99%*** (µsec)	***99.9%*** (µsec)
125k	***64***	***1***	124979	13	15	6	11	667	41	61
		2	124979	13	15	6	11	1005	41	61
	128	***1***	124979	13	15	5	11	238	39	60
		2	124979	13	16	9	11	11129	42	63
	256	***1***	124979	13	16	6	11	571	39	58
		2	124979	14	16	6	11	360	39	59
	512	***1***	124979	15	17	6	12	103	40	59
		2	124979	15	17	6	12	204	43	61
	1024	***1***	124979	17	20	7	14	139	44	62
		2	124979	16	20	7	14	273	45	62
270k	***64***	***1***	269955	16	19	8	10	885	44	78
		2	269955	17	19	8	11	961	44	78
	128	***1***	269955	19	36	164	11	8398	147	2989
		2	269955	19	36	160	11	8048	152	2992
	256	***1***	269955	24	36	76	12	4948	186	1377
		2	269955	23	35	72	11	2526	183	1261
	512	***1***	269955	32	35	15	12	262	83	129
		2	269955	31	34	13	12	297	76	101
	1024	***1***	269887	45	51	16	19	214	113	140
		2	269809	44	51	16	19	259	113	140
340k	***64***	***1***	337823	22	90	271	11	5547	1503	3294

		2	339943	22	84	327	11	9678	1123	5592
	128	1	339943	24	52	131	11	3273	487	1851
		2	339943	24	53	136	12	3477	489	1936
	256	1	339943	31	42	50	12	1308	245	712
		2	339943	30	40	49	12	1402	224	737
	512	1	339942	51	50	16	15	261	104	135
		2	339942	51	50	16	15	299	103	135
	1024	1	*							
		2	*							
650k	64	1	649890	40	58	60	16	1178	321	599
		2	649890	41	62	63	16	1022	339	610
	128	1	649891	62	68	33	17	607	175	257
		2	649890	62	69	33	18	589	172	256
	256	1	649843	68	72	11	26	260	110	130
		2	649838	68	72	11	32	337	111	134
	512	1	*							
		2	*							
	1024	1	*							
		2	*							
790k	64	1	789867	62	74	36	23	437	189	263
		2	789866	62	74	36	25	524	189	276
	128	1	789858	92	95	11	34	314	132	152
		2	789795	91	95	12	28	362	136	158
	256	1	*							
		2	*							
	512	1	*							
		2	*							
	1024	1	*							
		2	*							
870k	64	1	859843	134	135	15	55	379	177	198
		2	859844	135	136	15	62	349	176	200
	128	1	*							
		2	*							
	256	1	*							
		2	*							
	512	1	*							
		2	*							
	1024	1	*							
		2	*							

A “*” in the table indicates that data loss was reported at this rate.

4.4 3:3, 64-Byte Messages, Throughput-Optimized Config

4.4.1 OpenOnload

Path	*Run*	*Actual Rate* (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
1	*1*	619,894	104	107	23	46	738	186	236
1	*2*	619,895	101	104	22	37	750	178	234
2	*1*	559,905	98	5985	36059	41	317805	245851	308001
2	*2*	559,905	98	100	20	39	929	160	237
3	*1*	559,904	97	99	19	39	793	155	217
3	*2*	559,905	98	99	19	41	831	157	224

4.4.2 Kernel Stack

Path	*Run*	*Actual Rate* (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
1	*1*	799,854	181	179	30	68	436	241	263
1	*2*	799,860	180	179	30	69	428	240	264
2	*1*	794,928	195	196	28	70	907	263	285
2	*2*	793,517	196	197	28	69	920	267	298
3	*1*	798,081	169	171	19	55	858	221	239
3	*2*	796,933	198	198	28	100	892	265	295

4.5 3:3, 1024-Byte Messages, Throughput-Optimized Config

4.5.1 OpenOnload

Path	*Run*	*Actual Rate* (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
1	*1*	249,958	47	54	17	17	266	115	150
1	*2*	249,958	47	53	17	18	251	115	146
2	*1*	149,975	52	60	22	20	951	137	214
2	*2*	149,975	51	59	22	20	1063	135	206
3	*1*	149,975	51	59	21	21	983	132	200
3	*2*	149,975	51	58	20	21	983	128	171

4.5.2 Kernel Stack

Path	*Run*	*Actual Rate* (msg/sec)	*Median* (µsec)	*Mean* (µsec)	*StdDev* (µsec)	*Min* (µsec)	*Max* (µsec)	*99%* (µsec)	*99.9%* (µsec)
1	*1*	179,970	93	94	16	37	264	138	161
1	*2*	179,970	93	94	16	38	285	140	164
2	*1*	167,934	108	112	25	50	951	190	222
2	*2*	168,061	108	112	25	51	989	191	228
3	*1*	168,890	108	113	25	50	943	194	229
3	*2*	168,141	108	113	25	52	954	192	227

About STAC

The Securities Technology Analysis Center, or STAC, conducts private and public hands-on research into the latest technology stacks for capital markets firms and their vendors. STAC provides performance measurement services, advanced tools, and simulated trading environments in STAC Labs. Public STAC Reports, available for free at www.STACresearch.com, document the capability of specific software and hardware to handle key trading workloads such as real-time market data, analytics, and order execution.

STAC also facilitates the STAC Benchmark Council, an organization of leading trading firms and vendors that specify standard ways to measure the performance of trading solutions (see www.STACresearch.com/council).

To be notified when new STAC Reports like this one are issued, or to learn more about STAC, see our web site at www.STACresearch.com.