SOLARFLARE®

Improving Web Server Performance: Nginx Acceleration with Solarflare OpenOnload and Flareon 10/40GbE Server I/O Adapters

SolarflareWhitePaper

Executive Summary

The Solarflare Flareon™ SFN7002F 10GbE Server I/O Adapter with Solarflare OpenOnload® can deliver up to a 120% increase in Nginx application performance over the Intel Ethernet Converged Network Adapter X710. Similarly, the Solarflare Flareon SFN7142Q 40GbE Server Adapter is able to scale Nginx performance significantly, achieving nearly 40Gps line rate. These performance increases are accomplished by using the OpenOnload high-performance, open source, user-level networking stack. OpenOnload enables the Nginx application to achieve higher performance by bypassing the operating system's kernel and leveraging the advanced functionality of Solarflare server I/O adapters. At both 10GbE and 40GbE, with OpenOnload, Nginx performance is only limited by the bandwidth of the link.

When we look at how many CPU cores it takes to saturate a 10GbE link, OpenOnload needs two cores. In contrast, the Intel X710 with its kernel driver needs six. The Solarflare SFN7002F with OpenOnload is found therefore to be three times more efficient in its use of precious server compute resources than the Intel X710. OpenOnload unlocks the power of the CPU cores and increases Nginx HTTP connection rates. In short, the Nginx application running with Solarflare optimized network I/O has more CPU resources available that can be used for other applications or to scale to higher transaction rates per server, lowering operating costs.

Introduction: Scope and Purpose

Many Web, Cloud, and CDN (content delivery network) customers are looking to increase web server performance by upgrading and optimizing components of their Web platform including compute, storage, network, and the Nginx software stack itself. To support these efforts to improve Web performance, this paper shows the superior scalability and throughput capability of Solarflare 10GbE and 40GbE server I/O adapters for Nginx when combined with the OpenOnload high-performance network stack.

Nginx[1] is a high-performance HTTP server. Nginx (or its commercial version Nginx Plus) is deployed in many Web and content delivery networks (CDNs) for servers to support short-lived and long-lived HTTP connections. This paper investigates the specific use case where a web server handles many concurrent, short-lived requests for moderately small payloads, modelling the case of a typical website with sub-10 Kbyte request sizes, rather than one serving large files or streaming media.

OpenOnload

Solarflare OpenOnload is a Linux-based, open source, high-performance application accelerator that delivers low latency and improves application scalability. Because OpenOnload presents a standard BSD sockets API to Nginx, the application requires no modification. OpenOnload achieves performance improvements in part by performing network processing at user-level, bypassing the OS kernel entirely on the data path. Application and network performance is improved without sacrificing the security and multiplexing functions that the OS kernel normally provides. The experiments described in this paper use OpenOnload 201502-u1.

[1] *http://wiki.nginx.org/Main Retrieved 2015-03-17*

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

SolarflareWhitePaper

Two features of OpenOnload that are particularly relevant to the benchmarking performed in this paper are `SO_REUSEPORT` and *socket caching*. The `SO_REUSEPORT` socket option allows multiple TCP listening sockets to bind to the same IP address and port and to distribute incoming connection requests between these sockets. This feature can be exploited by applications to increase their scalability. Support for `SO_REUSEPORT` was added to OpenOnload in version 201405 and to Linux in kernel 3.9.

OpenOnload's implementation of `SO_REUSEPORT` is superior to that of the Linux kernel. Specifically, OpenOnload uses the Flareon adapter's hardware spreading as opposed to the kernel's software approach. OpenOnload also has the ability to instantiate independent, per process, per thread network stacks each having direct access to hardware. This eliminates lock contention between processes when sending to and receiving from the network stack.

Socket caching is a feature that reduces the processing overheads of establishing a TCP connection. Ordinarily, when an application accelerated by OpenOnload accepts a TCP connection, it must make a context switch into the kernel to allocate resources and to configure the hardware. Socket caching allows this state to be preserved when an accepted socket is closed and reused by a subsequent socket. This avoids the context switch and so reduces the total time taken for the socket to be accepted. This removes a significant bottleneck for applications with many short-lived sockets. Support for socket caching was added to OpenOnload in version 201502.

The Benchmark

Test Setup

This section gives an overview of the systems used to generate the results detailed in this paper. Full specifications are given in the Appendix 1.

Nginx Server

In each test, a single machine was used as the Nginx web server. This system had two six-core Haswell-architecture Intel Xeon E5-2620 v3 processors with hyper-threading running at 2.4 GHz, and 64 GB DDR4 RAM running at 1867 MHz.

The number of Nginx processes was varied in the tests up to the maximum number of physical cores in the system. These processes were single-threaded. They each bound a single listening socket at the same port with the `SO_REUSEPORT` socket option set, so that the requests were distributed amongst the server processes. The content served was static and 10,000 bytes in length and was stored in a RAM-backed file system.

Nginx 1.7.7 was used for the benchmark, together with a patch to enable `SO_REUSEPORT`[2].

Clients

Eight load-generating client machines were used. Each of these had a single four-core Sandy Bridge-architecture Intel Xeon processor running at 3.2 GHz, and 16 GB DDR3 RAM running at 1333 MHz. As with the server, these ran RHEL 7. Each system had a single Solarflare SFN6122F 10GbE network adapter.

The requests to the server were generated using the "ApacheBench" (`ab`)[3] tool . This was selected for the detail of its statistical reporting and its standing as a standard benchmarking application.

[2] *http://forum.nginx.org/read.php?29,241470 Retrieved 2015-03-18.*
[3] *http://httpd.apache.org/docs/2.2/programs/ab.html Retrieved 2015-03-17.*

SOLARFLARE®

sales@solarflare.com
US 1.949.581.6830 x2930
UK +44 (0)1223 477171
HK +852 2624-8868
www.solarflare.com

In each test, four `ab` instances were spawned on each client host, configured to attempt an equal number of concurrent connections varying according to the specific test. The sum of these attempted connections across all `ab` instances across all the client hosts is denoted in this paper by "concurrency". The `ab` instances were run without keep-alive, meaning that each `ab http` request established a new TCP connection.

The kernel networking stack handled the traffic on the client machines.

Connectivity

A high port density top of rack 10GbE switch was used to connect the clients and the server. This switch has forty-eight 10GbE SFP+ ports and four 40GbE QSFP ports. The clients and server were each connected to the switch on only one of the two ports of the dual-port network adapters.

The network interface on the server was configured with a single IP address, whereas sixteen IP addresses were configured on each client interface to avoid TCP port-space exhaustion.

Methodology

Given a set of hardware, the benchmark's goal was to maximize the request rate that Nginx can handle. The number of CPU cores available was artificially restricted in order to demonstrate the comparative efficiency of OpenOnload versus the Linux kernel. The paper presents results where there are *x* physical cores available for values of *x* from 1 to 6. These cores reside on a single package. The paper also gives results for *x* = 8, 10 and 12, and in these cases the cores are divided between the two packages. The division differs whether OpenOnload is used. In all cases hyper threading was enabled, which turned each physical core into 2 logical CPU cores.

Interrupts and Nginx processes were then affinitized to logical cores in a manner found to be optimal for the given configuration. The specific optimization settings depended on the specific adapter and whether OpenOnload was used (See **Tables 1 and 2**).

Without OpenOnload

In the case where OpenOnload was not used, no common configuration was found that obtained the best performance on all adapters in terms of requests per second. Therefore, two pinning configurations were used.

In the first configuration that delivered best performance for the Intel X710, each logical CPU was dedicated exclusively to either interrupt handling or application processing. Depending on the number of cores available, the method of workload allocation to each hyper thread differed.

For smaller number of cores, where the use case fits into single package, the load gets distributed in the following manner: on each core one logical CPU is dedicated to application processing, while the other CPU is dedicated to interrupt handling.

On the other hand, for larger number of cores, where Nginx process instances do not fit into a single package, the workload gets distributed differently. Half of the cores come from the NIC's local package and the other half come from the remote package. For this case, all the logical CPUs on the local package were dedicated to interrupt handling, while all the logical CPUs on the remote package were dedicated to application processing.

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

The second configuration benefits the Solarflare SFN7002F server adapter where the Nginx process instances do fit into a single package. In this configuration, both interrupts and application are simply pinned to the same hyper thread on the NIC's local package.

Experiments have shown that the above set up methodology gave the best results across a broad parameter space.

Core Allocation & Pinning Configuration 1

I = Hyper thread assignment is IRQ
A = Hyper thread assignment is application
B = Both
Blank = No allocation

3 Cores with Kernel Stack (only 3 cores on NIC local package used)

Package\Core	0	1	2	3	4	5
Local	IA	IA	IA			
Remote						

6 Cores with Kernel Stack (only 3 cores on NIC local package used)

Package\Core	0	1	2	3	4	5
Local	IA	IA	IA	IA	IA	IA
Remote						

8 Cores with Kernel Stack (only 3 cores on NIC local package used)

Package\Core	0	1	2	3	4	5
Local	II	II	II	II		
Remote	AA	AA	AA	AA		

Continued next page

SolarflareWhitePaper

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

Core Allocation & Pinning Configuration 2	I = Hyper thread assignment is IRQ A = Hyper thread assignment is application B = Both Blank = No allocation

3 Cores with Kernel Stack (only 3 cores on NIC local package used)

Package\Core	0	1	2	3	4	5
Local	I/AI/A	I/AI/A	I/AI/A			
Remote						

6 Cores with Kernel Stack (only 3 cores on NIC local package used)

Package\Core	0	1	2	3	4	5
Local	I/AI/A	I/AI/A	I/AI/A	I/AI/A	I/AI/A	I/AI/A
Remote						

Table 1. Examples of core allocation and pinning configuration.
Note: When eight or more cores are in use pinning is exactly the same as with Configuration 1.

With OpenOnload

In the case where OpenOnload was used, the Nginx processes were accelerated with OpenOnload configured in *spinning* mode. This means that application threads using OpenOnload can busy-wait when making network stack calls for a configurable duration, before sleeping and then being re-scheduled as the result of an interrupt when work arrives. (100ms in the experiments conducted for this paper). In practice this means that the application context performs most tasks that would otherwise have been done in the interrupt context. As a consequence, OpenOnload will generate very little interrupt load, eliminating the need to dedicate (logical) CPU cores to interrupt handling. Therefore, for benchmarking, logical cores were assigned on the basis of using NIC local package logical cores first and then using the NIC remote package cores. Interrupt threads were allocated per NIC local core.

Core Allocation & Pinning Configuration	I = Hyper thread assignment is IRQ A = Hyper thread assignment is application B = Both Blank = No allocation

3 Cores with OpenOnload

Package\Core	0	1	2	3	4	5
Local	BB	BB	BB			
Remote						

Continued next page

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

SolarflareWhitePaper

Core Allocation & Pinning Configuration						
I = Hyper thread assignment is IRQ A = Hyper thread assignment is application B = Both Blank = No allocation						
3 Cores with OpenOnload						
Package\Core	0	1	2	3	4	5
Local	BB	BB	BB			
Remote						
6 Cores with OpenOnload						
Package\Core	0	1	2	3	4	5
Local	BB	BB	BB	BB	BB	BB
Remote						
6 Cores with OpenOnload						
Package\Core	0	1	2	3	4	5
Local	BB	BB	BB	BB	BB	BB
Remote	AA	AA				

Table 2. Examples of core allocation and pinning configuration.

Benchmarking Parameters

In order to model a realistic web server scenario, it was necessary to run the `ab` clients with a fair level of concurrency. If the concurrency level was set too high, the server load increased beyond its capacity and the aim of the benchmark to assess processing efficiency became confounded by other factors, e.g. TCP connection retries, etc. It was decided to run the tests at a concurrency of 2048. This figure was reached by running preliminary experiments at a variety of concurrencies and selecting the maximum number of concurrent connections where the time to service a request at the 99th percentile was below 1000 ms for all of the adapters under test.

Results

Benchmark experiments were conducted on 10GbE Intel adapters and Solarflare 10GbE and 40GbE adapters. **Figure 1** (next page) shows the performance results of the SFN7002F 10GbE adapter with and without OpenOnload versus the Intel X710 with kernel driver.

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

SolarflareWhitePaper

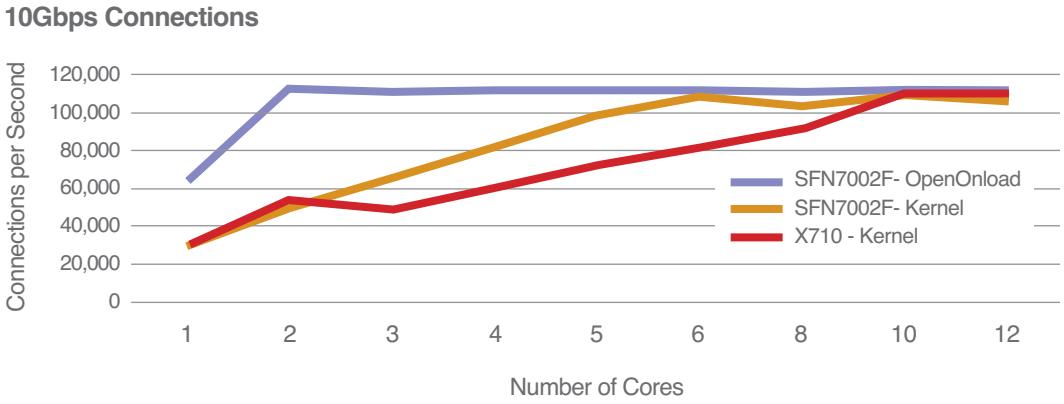

Figure 1. Solarflare vs. Intel Nginx Performance: Cores and Connections per Second.

Figure 1 plots Nginx performance on the y-axis versus the allocated number of cores on the x-axis. Performance is represented as a request rate, in this case, connections per second. The figure shows that as the number of cores is increased (i.e. additional Nginx instances are run) the request rate increases until reaching a ceiling of 111.6 thousand connections per second.

The results show that the Solarflare SFN7002F with OpenOnload delivers a maximum 120% increase in Nginx performance over the Intel X710 with Flow Director. With Flow Director off, OpenOnload yields a 128% boost over the Intel X710. When we look at how many CPU cores it takes to saturate a 10GbE link, OpenOnload needs two cores. In contrast, the Intel X710 adapter (and the Solarflare adapter with its kernel driver) needs six or more cores. OpenOnload is therefore found to be three times more efficient than the kernel in its use of CPU resources.

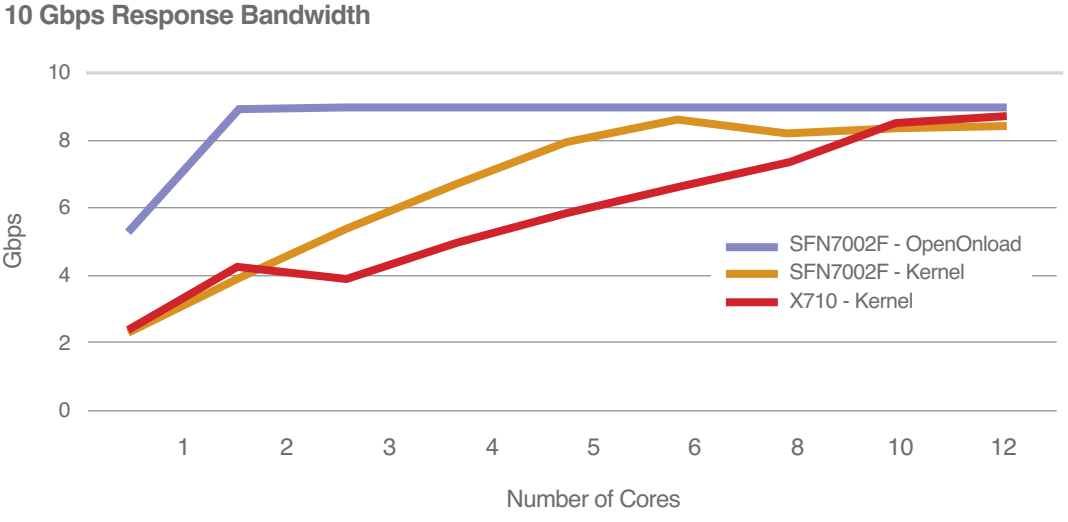

Figure 2. Solarflare vs. Intel Nginx Performance: Cores and Response Bandwidth (Data Rate)

Beyond 10 Gbps: Scaling with OpenOnload and Solarflare Flareon Ultra 40GbE Adapters

The results in **Figure 2** above demonstrate that Nginx with OpenOnload can saturate a single 10 Gbps link with the use of very few CPU cores. In order to demonstrate that link bandwidth is truly the bottleneck, an experiment was conducted with a Solarflare Flareon® Ultra SFN7142Q dual-port 40 Gbps adapter.

SOLARFLARE®

sales@solarflare.com
US 1.949.581.6830 x2930
UK +44 (0)1223 477171
HK +852 2624-8868
www.solarflare.com

SolarflareWhitePaper

40 Gbps Connections

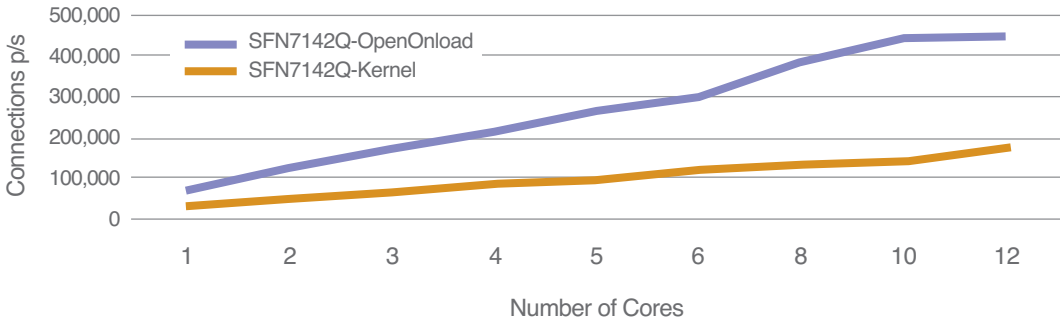

Figure 3. 40GbE Nginx Performance: Cores and Connection Rate.

40 Gbps Response Bandwidth

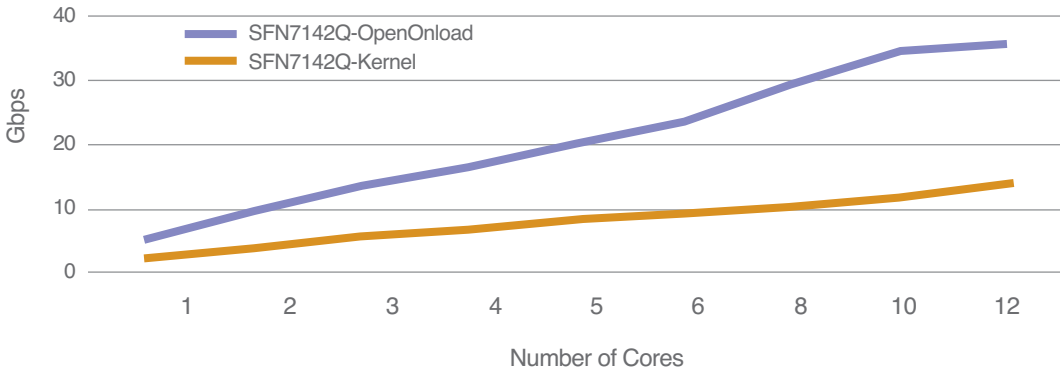

Figure 4. 10GbE Nginx Performance: Cores and Response Bandwidth (Data Rate).

Future Work

The next phase of Solarflare's Nginx testing will experiment with different payload sizes, especially for long-lived HTTP connections such as those used by CDNs and IP video networks. A subsequent phase will also include examination of Nginx and Nginx Plus as load balancers. Solarflare will also test the performance of four 10GbE ports and two 40GbE ports with OpenOnload and compare and contrast Solarflare with the Intel XL710 10/40GbE server adapter.

Conclusions

With OpenOnload, an industry standard server with a given CPU resource allocation running Nginx can achieve up to 120% higher throughput than the Intel X710. The Solarflare SFN7002F with OpenOnload is three times more efficient than the Intel X710 in its use of precious and expensive CPU resources. Moreover, the SFN7002F can saturate a 10GbE link with two cores while the SFN7142Q adapter can saturate a 40GbE link with 10,000 byte payload requests and the use of 10 cores. With OpenOnload, Nginx performance is only limited by the bandwidth of the link.

Bottom line: Nginx applications running with Solarflare have more CPU resources available that can be used for other applications or to scale to higher transaction rates per server, thus lowering operating costs.

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

SolarflareWhitePaper

Appendix I. Test Setup Specifications

Server

- One Dell R630 server
- Two Intel Xeon E5-2620 v3 CPUs at 2.40 GHz (Haswell; six cores; hyper threading)
- 64 GB DDR4 SD-RAM at 1867 MHz
- Red Hat Enterprise Linux 7.0, kernel version: 3.10.0-123.el7.x86_64
- Nginx 1.7.7 (patched for `SO_REUSEPORT`: see above)
- Network adapters:
  - Solarflare SFN7002F, ultra-low latency firmware
  - Solarflare SFN7142Q, ultra-low latency firmware
  - Intel X710
- Network adapter driver versions:
  - Solarflare Onload: OpenOnload-201502-u1/ v4.4.1.1017
  - Intel X710: i40e 1.2.37

Clients

- Eight Dell R210 servers.
- Single Intel Xeon E3-1230 CPU at 3.20 GHz (Sandy Bridge; four cores, no hyper threading)
- 16 GB DDR3 SD-RAM at 1333 MHz
- ApacheBench 2.3 (patched: see above)
- Network adapters: SFN6122F, Driver version OpenOnload-201502-u1

Appendix II. Server Tuning

Common Settings

- CPU frequency scaling: disabled
- Adaptive interrupt moderation: disabled
- Interrupt moderation: 60
- Large receive offload: enabled
- TCP segmentation offload: enabled
- Flow Steering: disabled on all NICs unless indicated otherwise
- Modules/services disabled: Network-manager iptables, ebtables, conntrack, irqbalance
- Sysctl values:

```
fs.file-max = 4000000
fs.nr_open = 4000000
net.ipv4.tcp_fin_timeout = 0
net.core.somaxconn = 65535
net.core.netdev_max_backlog = 131000
net.ipv4.ip_local_port_range = 2000 65535
net.ipv4.tcp_rfc1337 = 1
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.tcp_max_tw_buckets = 5880000
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
```

Continued next page

SOLARFLARE®

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

Common Settings - continued

• Sysctl values:

```
net.ipv4.tcp_max_orphans = 1000000
net.ipv4.neigh.default.gc_thresh1 = 1024
net.ipv4.neigh.default.gc_thresh2 = 2048
net.ipv4.neigh.default.gc_thresh3 = 4096
kernel.shmmni = 16000
kernel.nmi_watchdog = 0
kernel.perf_event_paranoid = -1
kernel.kptr_restrict = 0
net.ipv4.tcp_mem = 48000000 48000000 48000000
net.ipv4.tcp_rmem = 8192 8192 16777216
net.ipv4.tcp_wmem = 4096 4096 16777216
```

Solarflare SFN7xxx Series Server I/O Adapters

Firmware version: v4.4.2.1011

Firmware variant: `ultra-low-latency`

OpenOnload-Specific Parameters

```
EF_PIO=0
EF_TX_PUSH=0
EF_USE_HUGE_PAGES=1
EF_ACCEPTQ_MIN_BACKLOG=65535
EF_EPOLL_SPIN=1
EF_POLL_SPIN=1
EF_POLL_FAST_USEC=1000
EF_EPOLL_MT_SAFE=1
EF_RXQ_SIZE=512..4096 depending on nginx threads used
EF_MIN_FREE_PACKETS=500000..100000 depending on number of nginx
threads used
```

Solarflare-Specific

```
Kernel module options:
rx_copybreak=256
rx_recycle_ring_size=512
tx_push_max_fill=0
```

Appendix III. ApacheBench modification

To overcome some of ApacheBench limitations it sources have been modified to achieve number of features:

• Allow synchronizing multiple `ab` instances.

• Enable use of array of ip addresses per `ab` instance.

• Do not stop despite connection failures.

• Preset expected response size.

• Provide number of successful requests.

sales@solarflare.com

US 1.949.581.6830 x2930

UK +44 (0)1223 477171

HK +852 2624-8868

www.solarflare.com

SF-114312-CD Issue 5
NX_WP_091715