Cloud Onload® NGINX Web Server Cookbook

SF-122073-CD

Issue 2

Table of Contents

1 Introduction

This chapter introduces you to this document. See:

1.1 About this document

This document is the *NGINX Web Server Cookbook* for Cloud Onload. It gives procedures for technical staff to configure and run tests, to benchmark NGINX Plus as a web server utilizing Solarflare's Cloud Onload and Solarflare NICs.

This document contains the following chapters:

- Introduction on page 1 (this chapter) introduces you to this document.
- Overview on page 3 gives an overviews of the software distributions used for this benchmarking.
- Summary of benchmarking on page 6 summarizes how the performance of NGINX Plus has been benchmarked, both with and without Cloud Onload, to determine what benefits might be seen.
- Installation and configuration on page 9 describes how to install and configure the software distributions used for this benchmarking.
- Evaluation on page 12 describes how the performance of the test system is evaluated.
- Benchmark results on page 29 presents the benchmark results that are achieved.

1.2 Intended audience

The intended audience for this *NGINX Web Server Cookbook* are:

- software installation and configuration engineers responsible for commissioning and evaluating this system
- system administrators responsible for subsequently deploying this system for production use.

1.3 Registration and support

Support is available from support@solarflare.com.

1.4 Download access

Cloud Onload can be downloaded from: https://support.solarflare.com/.

Solarflare drivers, utilities packages, application software packages and user documentation can be downloaded from: https://support.solarflare.com/.

Please contact your Solarflare sales channel to obtain download site access.

1.5 Further reading

For advice on tuning the performance of Solarflare network adapters, see the following:

- *Solarflare Server Adapter User Guide* (SF-103837-CD).
  This is available from: https://support.solarflare.com/.

For more information about Cloud Onload, see the following:

- *Onload User Guide* (SF-104474-CD).
  This is available from: https://support.solarflare.com/.

2 Overview

This chapter gives an overview of the software distributions used for this benchmarking. See:

2.1 NGINX Plus overview

Open source NGINX [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server.

NGINX Plus is a software load balancer, web server, and content cache built on top of open source NGINX. NGINX Plus has exclusive enterprise-grade features beyond what's available in the open source offering, including session persistence, configuration via API, and active health checks.

NGINX Plus is heavily network dependent by design, so its performance can be significantly improved through enhancements to the underlying networking layer.

2.2 Wrk overview

Wrk is a modern HTTP benchmarking tool capable of generating significant load when run on a single multi-core CPU. It combines a multithreaded design with scalable event notification systems such as epoll and kqueue.

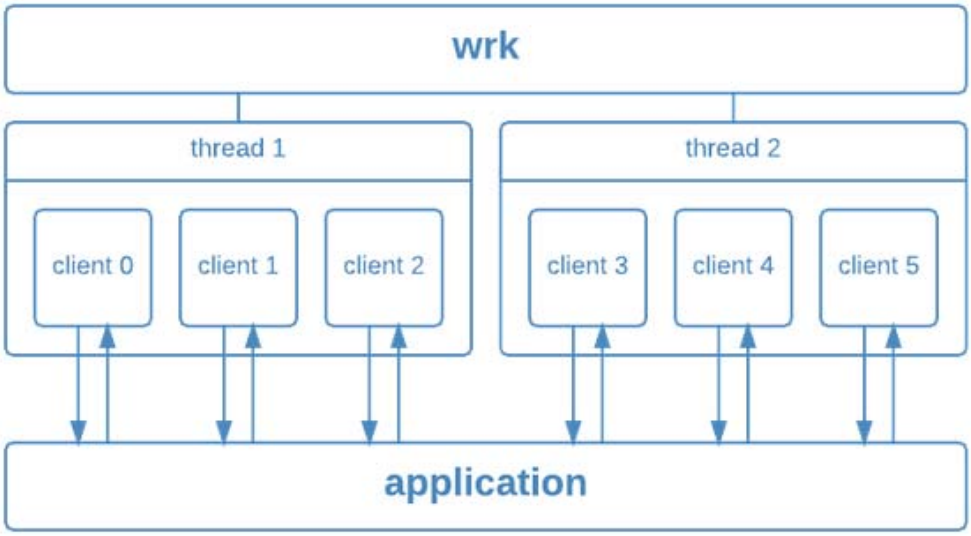

Figure 1: Wrk architecture

2.3 Cloud Onload overview

Cloud Onload is a high performance network stack from Solarflare (https://www.solarflare.com/) that dramatically reduces latency, improves CPU utilization, eliminates jitter, and increases both message rates and bandwidth. Cloud Onload runs on Linux and supports the TCP network protocol with a POSIX compliant sockets API and requires no application modifications to use. Cloud Onload achieves performance improvements in part by performing network processing at user-level, bypassing the OS kernel entirely on the data path.

Cloud Onload is a shared library implementation of TCP, which is dynamically linked into the address space of the application. Using Solarflare network adapters, Cloud Onload is granted direct (but safe) access to the network. The result is that the application can transmit and receive data directly to and from the network, without any involvement of the operating system. This technique is known as “kernel bypass”.

When an application is accelerated using Cloud Onload it sends or receives data without access to the operating system, and it can directly access a partition on the network adapter.

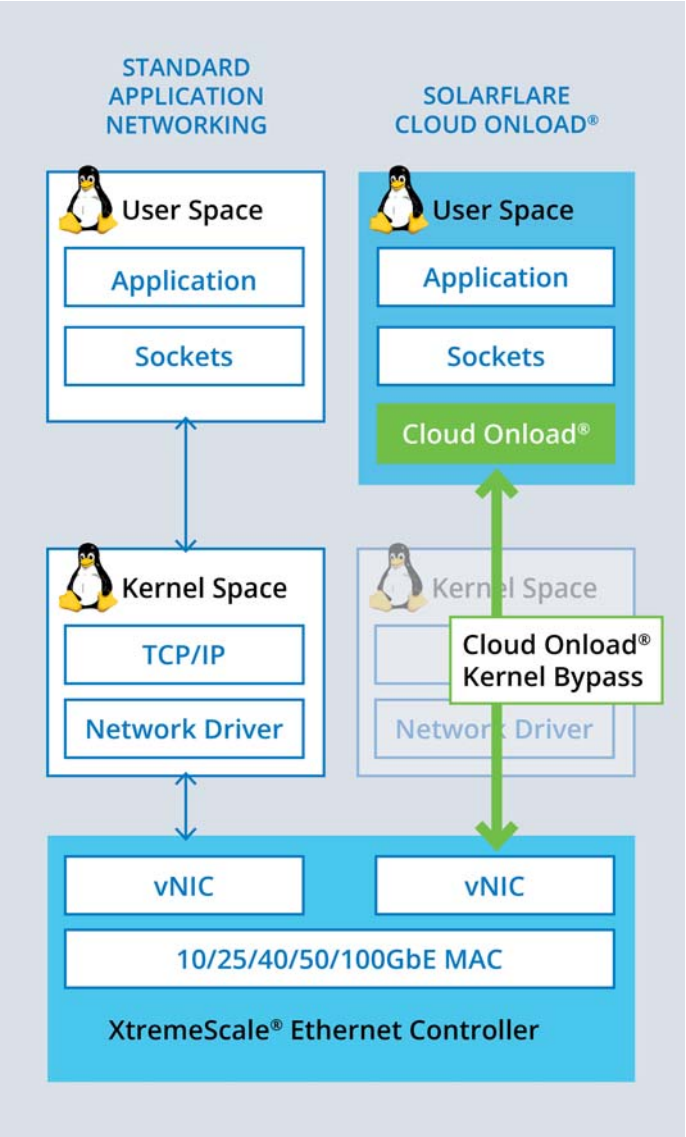

Figure 2: Cloud Onload architecture

3 Summary of benchmarking

This chapter summarizes how the performance of NGINX Plus as a web server has been benchmarked, both with and without Cloud Onload, to determine what benefits might be seen. See:

- Architecture for NGINX Plus benchmarking on page 7
- NGINX Plus benchmarking process on page 8.

3.1 Architecture for NGINX Plus benchmarking

Benchmarking was performed with two Dell R740 servers, with the following specification:

Server	Dell R740XD
Memory	192GB (12 × 16384 MB)
NICs	SFN8522 (dual port 10G) X2522-25G (dual port 25G) X2541 (single port 100G)
CPU	sfocr740a (used for `wrk`): 2 × Intel® Xeon® Platinum 8153 CPU @ 2.00GHz sfocr740b (used for `nginx`): Intel® Xeon® Gold 5120 CPU @ 2.20GHz
OS	Red Hat Enterprise Linux Server release 7.5 (Maipo)
Software	NGINX 1.15.7 (NGINX Plus r17) wrk 4.1.0

Each server is configured to leave as many CPUs as possible available for the application being benchmarked.

All high-volume test traffic is routed through a dedicated switch that provides 10, 25, and 100GbE ports.

This enables testing at three different network speeds (10GbE, 25GbE and 100GbE) to determine when applications become bottlenecked as a result of network traffic.

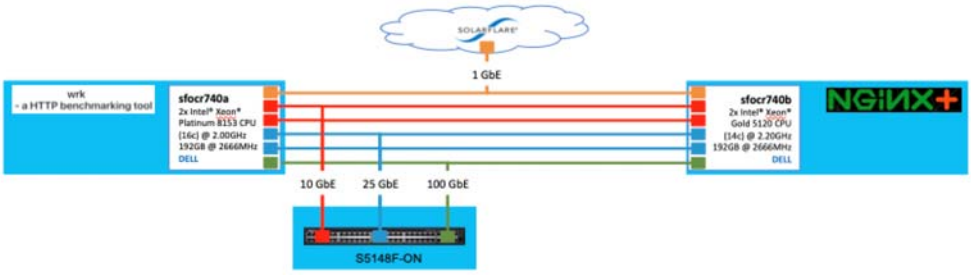

Figure 3: Architecture for NGINX Plus benchmarking

3.2 NGINX Plus benchmarking process

These are the high-level steps we followed to complete benchmarking with NGINX Plus:

- Install and test NGINX Plus on one server (sfoc740b).
- Install wrk on the other server (sfoc740a).
- Start an NGINX web server on one node (sfoc740b).

  The first iteration of the test uses a single worker process.
- Start `wrk` on the other node (sfoc740a).

  All iterations of the test use the same configuration for consistency:
  - One `wrk` process is assigned to each CPU.

    For the server used (sfoc740a), this is 32 `wrk` processes
  - Each `wrk` process is accelerated by Cloud Onload, to maximize the throughput of each connection going to the NGINX Plus web server.
- Record the response rate of the NGINX server, as the number of requests per second.
- Increment the number of NGINX worker processes, and repeat the test.

  Continue doing this until one NGINX worker process is assigned to each CPU. For the server used (sfoc740b), this is 28 processes.

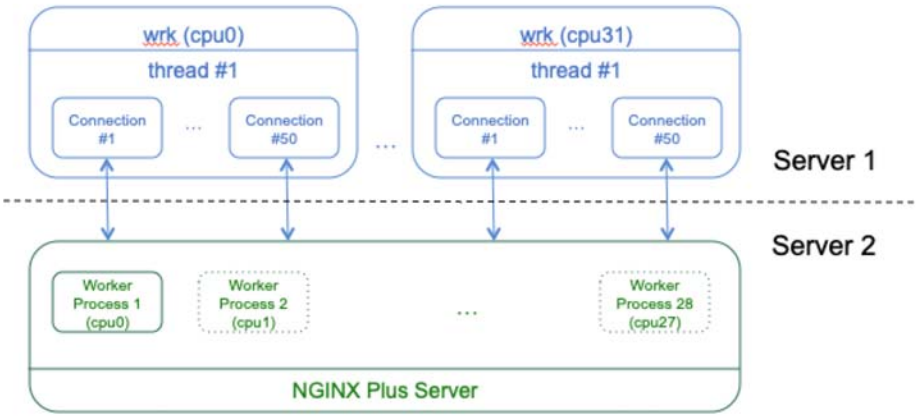

Figure 4: NGINX Plus software usage

- Repeat the test across all interfaces available on the server.
- Repeat all tests, accelerating NGINX Plus with Cloud Onload.

These steps are detailed in the remaining chapters of this *Cookbook*.

4 Installation and configuration

This chapter describes how to install and configure the software distributions used for this benchmarking. See:

- Installing NGINX Plus on page 9
- Installing wrk on page 11.

4.1 Installing NGINX Plus

This section describes how to install and configure NGINX Plus.

Installation

NOTE: For a reference description of how to install NGINX Plus, see https://cs.nginx.com/repo_setup.

In summary:

1. If you already have old NGINX packages in your system, back up your configs and logs:

   ```
   # cp -a /etc/nginx /etc/nginx-plus-backup
   # cp -a /var/log/nginx /var/log/nginx-plus-backup
   ```

2. Create the `/etc/ssl/nginx/` directory:

   ```
   # mkdir -p /etc/ssl/nginx
   ```

3. Log in to NGINX Customer Portal and download the following two files:

   - `nginx-repo.key`
   - `nginx-repo.crt`

4. Copy the above two files to the RHEL/CentOS/Oracle Linux server into `/etc/ssl/nginx/` directory. Use your SCP client or other secure file transfer tools.

   ```
   # cp <path>/nginx-repo.* /etc/ssl/nginx/.
   ```

5. Install prerequisite packages:

   ```
   # yum install ca-certificates
   ```

6. Add the NGINX Plus repository by downloading the file `nginx-plus-7.4.repo` to `/etc/yum.repos.d`:

   ```
   # wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/nginx-plus-7.4.repo
   ```

7. Install the NGINX Plus package:

   ```
   # yum install nginx-plus
   ```

8 Check the NGINX binary version to ensure that you have NGINX Plus installed correctly:

```
# nginx -v
nginx version: nginx/1.15.7 (nginx-plus-r17)
```

9 Start NGINX:

```
# systemctl start nginx
```

or just:

```
# nginx
```

10 Verify access to Web Server

Configuration

The NGINX configuration file is `/etc/nginx/nginx.conf`.

To define the number of worker processes that will get instantiated by NGINX, modify the `worker_processes` variable.

```
# vi /etc/nginx/nginx.conf
…
user  nginx;
#worker_processes  auto;
worker_processes  28;
```

4.2 Installing wrk

This section describes how to install and configure `wrk`.

Installation

NOTE: For a reference description of how to install `wrk`, see: https://github.com/wg/wrk/wiki/Installing-Wrk-on-Linux.

In summary:

1. If the build tools are not already installed, install them:

```
# yum groupinstall 'Development Tools'
```

2. If the OpenSSL dev libs are not already installed, install them:

```
# yum install -y openssl-devel
```

3. If git is not already installed, install it:

```
# yum install -y git
```

4. Create a directory to hold wrk:

```
# mkdir -p Onload_Testing/WRK
# cd Onload_Testing/WRK
```

5. Use git to download wrk:

```
# git clone https://github.com/wg/wrk.git wrk
```

6. Build wrk:

```
# cd wrk
# make
```

7. Copy the wrk executable to a location on your PATH. For example:

```
# cp wrk /usr/local/bin
```

5 Evaluation

This chapter describes how the performance of the test system is evaluated. See:

5.1 Configuring NGINX Plus for benchmarking

Operating System recommendations

To configure NGINX we first set up our Linux environment.

- Increase the local port range:

  ```
  sysctl -w net.ipv4.ip_local_port_range='9000 65000';
  ```

  This is so that the server can open lots of outgoing network connections.

- Set HugePages:

  ```
  sysctl -w vm.nr_hugepages=10000
  ```

  HugePages is a method to have larger pages, and is useful for working with very large memory.

- Increase the maximum number of open files by setting new limits for open files and file descriptors:

  ```
  sysctl -w fs.file-max=8388608;
  ulimit -n 8388608;
  ```

  Many application such as database or web server need a large amount of open files.

- Increase the number of files that a process can open.

  sysctl -w fs.nr_open=8388608;

- To start NGINX Plus with these settings, run the start script (`start_nginx`):

```
# cat start_nginx
#!/bin/bash

killall nginx

#  set -x;
  sysctl -w net.ipv4.ip_local_port_range='9000 65000';
  sysctl -w vm.nr_hugepages=10000;
  sysctl -w fs.file-max=8388608;
  sysctl -w fs.nr_open=8388608;
  ulimit -n 8388608;

# Start Nginx

nginx
```

Set up filesystem for static NGINX Plus files for benchmarking performance

NGINX Plus Web Server Configuration - The configuration below was used on the NGINX Plus Web Server. It serves static files from `/var/www/html/`, as configured by the `root` directive. The static files were generated using `dd`; this example creates a 1 KB file of zeroes:

```
dd if=/dev/zero of=1kb.bin bs=1KB count=1
```

The files used range from 0KB to 100MB files and reside in the `/var/www/html` directory.

```
# ls -lh /var/www/html/
total 107M
-rw-r--r-- 1 root root    0 Mar 27 21:14 0kb.bin
-rw-r--r-- 1 root root  98K Mar 27 19:43 100kb.bin
-rw-r--r-- 1 root root  96M Mar 27 19:44 100Mb.bin
-rw-r--r-- 1 root root 9.8K Mar 27 19:40 10kb.bin
-rw-r--r-- 1 root root 9.6M Mar 27 19:44 10Mb.bin
-rw-r--r-- 1 root root 1000 Mar 27 19:40 1kb.bin
-rw-r--r-- 1 root root 977K Mar 27 19:43 1Mb.bin
```

Filesystem where content is located needs to be mounted with `noatime` (otherwise each access generate disk write)

```
# mount -t tmpfs -o size=512m,noatime tmpfs <...>/html
```

```
# ls /root/Onload_Testing/NGINXPlus/nginx_webserver/html/
0kb.bin  100kb.bin  100Mb.bin  10kb.bin  10Mb.bin  1kb.bin  1Mb.bin
```

```
# cat mount_tmpfs
mount -t tmpfs -o size=512m,noatime tmpfs /var/www/html
```

```
# cp -a /root/Onload_Testing/NGINXPlus/nginx_webserver/html/*
/var/www/html/.
```

The nginx.conf configuration file

In order to start up X number of NGINX workers we have 28 different nginx.conf files. Each file represents the number of worker processes that will be run.

- nginx-webserver_X.conf list

```
# ls -ls nginx-webserver_*.conf
4 -rw-r--r-- 1 root root 1027 Apr  4 19:51 nginx-webserver_1.conf
…
4 -rw-r--r-- 1 root root 1488 Apr  4 20:04 nginx-webserver_28.conf
```

The key changes to the nginx.conf files are show below.

- Choose the number of worker processes

```
worker_processes 28;
```

- Enable worker CPU affinity, Binds worker processes to the sets of CPUs.

```
worker_cpu_affinity 01 010 0100 01000 …
```

- Enable file descriptor cache

```
open_file_cache max=1000 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_errors off;
```

- Increase number of open files for worker processes

  Changes the limit on the maximum number of open files (RLIMIT_NOFILE) for worker processes. Used to increase the limit without restarting the main process.

```
worker_rlimit_nofile 8388608;
```

- Enable reuseport directive

  This enables the kernel to have more socket listeners for each socket (ip:port).

  Without it, when new connection arrives, kernel notified all nginx workers about it and all of them try to accept it.

  With this option enabled, each worker has its own listening socket and on each new connection, kernel chooses one of them which will receive it - so there is no contention.

```
listen 80 reuseport;
```

- nginx.conf file for 28 worker processes

```
# cat nginx-webserver_28.conf
user                nginx;
worker_processes 28;
#worker_processes     auto;
#worker_rlimit_nofile 10240;
worker_rlimit_nofile   8388608;
worker_cpu_affinity 01 010 0100 01000 010000 0100000 01000000
010000000 0100000000 01000000000 010000000000 0100000000000
0100000000000 01000000000000 010000000000000 0100000000000000
0100000000000000 01000000000000000 010000000000000000
0100000000000000000 01000000000000000000 0100000000000000000000
0100000000000000000000 01000000000000000000000
0100000000000000000000000 0100000000000000000000000
0100000000000000000000000000 01000000000000000000000000;

pid                 /var/run/nginx_28.pid;

events {
#    worker_connections 10240;
    worker_connections 200000;
    accept_mutex       off;
    multi_accept       off;
}

http {
    access_log   off;
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local]
"$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"
"$ssl_cipher" '
                    '"$ssl_protocol" ';

    sendfile on;

    keepalive_timeout  300s;
    keepalive_requests 1000000;

    server {
#########
        listen 80 reuseport;
#########
        root /var/www/html;

#########
        open_file_cache max=1000 inactive=20s;
        open_file_cache_valid 30s;
        open_file_cache_errors off;
#########
    }
}
```

5.2 Starting NGINX Plus for kernel benchmarking

Testing methodology

We tested the performance of NGINX Plus web server with different numbers of CPUs. One NGINX Plus worker process consumes a single CPU, so to measure the performance of different numbers of CPUs we varied the number of NGINX worker processes, repeating the tests with one worker processes, two, four, eight, sixteen and the maximum number of CPUs on our server, 28.

NOTE: To set the number of NGINX worker processes manually, use the worker_processes directive. The default value is auto, which tells NGINX Plus to detect the number of CPUs and run one worker process per CPU.

Starting nginx

The following start_nginx file is called by the client node sfocr740a.

- go_namespace (run on client)

```
# less go_namespace
#!/bin/bash
for i in `seq 1 28`
do
                ssh sfocr740b   \
"/root/Onload_Testing/NGINXPlus/start_nginx $i"
..
done
```

- start_nginx

```
# cat start_nginx
#!/bin/bash
killall -9 nginx
NPROCS=$1

sysctl -w net.ipv4.ip_local_port_range='9000 65000';
sysctl -w vm.nr_hugepages=10000;
sysctl -w fs.file-max=8388608;
sysctl -w fs.nr_open=8388608;
ulimit -n 8388608;

nginx -c /root/Onload_Testing/NGINXPlus/nginx_webserver/nginx-
webserver_"$NPROCS".conf
```

5.3 Running wrk for kernel benchmarking

Performance metrics

We measured the following metrics:

- Requests per second (RPS)

  Measures the ability to process HTTP requests. In our tests, each client sends requests for a 0KB, 1KB, 10KB, and 100KB files over a keepalive connection.

- Transactions per second (TPS)

  Measures the ability to process new connections. In our tests, each client sends a series of HTTP requests, each on a new connection. The Web Server sends back a 0 byte response for each request.

- Throughput

  Measures the throughput that NGINX Plus can sustain when serving 100KB - 100MB files over HTTP.

wrk options

The following options are available for running `wrk`.

Table 1: Redis-benchmark options

Option	Description	Default value
`-t` `--threads`	Total number of threads to use.	2
`-c` `--connections`	Total number of HTTP connections to keep open. Each thread handles *N* connections, where *N* = (total connections / total threads).	10
`-d` `--duration`	Duration of the test, e.g. 2s, 2m, 2h.	10s
`-s` `---script`	LuaJIT script. See the `SCRIPTING` file in the source code.	—
`-H` `--header`	HTTP header to add to request, e.g. `"User-Agent: wrk"`	—
`--latency`	Print detailed latency statistics.	False
`--timeout`	Record a timeout if a response is not received within this amount of time.	Socket timeout

Of the above options, the following have been changed:

- `-t`: Threads.

  A value of 1 was used for testing.

- `-c`: Connections.

  A value of 50 was used for testing.

- `-d`: Duration.

  A value of 180s was used for testing.

- `-H`: Header.

  When used for TPS measurements, -H 'Connection: close' was used to ensure that a new connection is always created.

Example run and output of wrk

```
# ./wrk -t 1 -c 50 -d 180s -H 'Connection: close' http://192.168.105.35/0kb.bin
Running 3m test @ http://192.168.105.35/0kb.bin
  1 threads and 50 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    22.08ms   63.83ms   1.61s    95.43%
    Req/Sec     1.40k     0.91k   33.39k    98.55%
  250247 requests in 3.00m, 58.23MB read
Requests/sec:   1390.17
Transfer/sec:    .33125MB
```

Onloading wrk (via Cloud Onload + namespaces)

In order to saturate the NGINX Plus server worker processes over the various NIC speeds, the number of wrk thread connections needs to be accelerated. This can be done by running multiple wrk threads on multiple client servers, or using Cloud Onload to accelerate the individual connections per thread. We use Cloud Onload plus network namespace to accelerate the individual connections per thread.

A network namespace is logically another copy of the network stack, with its own routes, firewall rules, and network devices. By default, a process inherits its network namespace from its parent. Initially all the processes share the same default network namespace from the init process.

We pin a single wrk process to a single CPU and run wrk via Cloud Onload using namespaces:

Example wrk run w/ Cloud Onload + namespace + taskset

```
EF_MAX_ENDPOINTS=400000 \
EF_UL_EPOLL=3 \
EF_DYNAMIC_ACK_THRESH=4 \
EF_TCP_TCONST_MSL=1 \
EF_SCALABLE_FILTERS=any=rss:active:passive \
EF_TCP_SHARED_LOCAL_PORTS_MAX=570000 CI_TP_LOG=7 \
EF_EPOLL_SPIN=1 \
EF_LOAD_ENV=1 \
EF_USE_HUGE_PAGES=2 \
EF_ACCEPTQ_MIN_BACKLOG=400 \
EF_CLUSTER_RESTART=1 \
EF_TCP_FIN_TIMEOUT=15 \
EF_MAX_PACKETS=205000 \
EF_TCP_SYNRECV_MAX=90000 \
EF_TCP_BACKLOG_MAX=400 \
EF_TCP_INITIAL_CWND=14600 \
EF_TCP_SHARED_LOCAL_PORTS_REUSE_FAST=1 \
EF_PIO=0 \
EF_SLEEP_SPIN_USEC=50 LD_PRELOAD=libonload.so \
EF_CLUSTER_SIZE=1 \
EF_POLL_USEC=100000 \
EF_UDP=0 \
EF_SOCKET_CACHE_MAX=40000 \
EF_FDTABLE_SIZE=8388608 \
EF_TCP_SHARED_LOCAL_PORTS_PER_IP_MAX=570000 \
EF_TCP_SHARED_LOCAL_PORTS_NO_FALLBACK=1 \
EF_TAIL_DROP_PROBE=1 \
EF_EPOLL_MT_SAFE=1 \
EF_HIGH_THROUGHPUT_MODE=1 \
EF_TCP_SHARED_LOCAL_PORTS=1000 \
EF_LOG_VIA_IOCTL=1 \
EF_TCP_SHARED_LOCAL_PORTS_PER_IP=1 \
EF_NO_FAIL=1 \
EF_CLUSTER_NAME=load \
EF_MIN_FREE_PACKETS=50000 \
EF_TX_PUSH=0 \
EF_SCALABLE_FILTERS_ENABLE=1 \
taskset -c $i ip netns exec net$i ./wrk -t $THREAD -c $CONNECTIONS \
        -d $TIME -H 'Connection: close' http://$IPADDR/0kb.bin \
        >> results/wrk_"$TEST"_"$INTERFACE"_"$i" &
```

Prepare namespace on client node (wrk node)

For each NIC IP prepare namespaces and macvlan interfaces for each one with an IP address

```
[root@sfocr740a WebServer]# cat prepare_namespaces_p4p1
#!/bin/bash

for i in $(seq 0 31); do
  ip netns add net$i
  # p4p1 is a SFC interface
  ip link add link p4p1 name mvl type macvlan
  ip link set netns net$i dev mvl
  # assuming 192.168.105.0/24 network is used for tests
  ip netns exec net$i ifconfig mvl 192.168.105.1$i/24 up
  ip netns exec net$i ifconfig lo up
done
```

Requests per second (RPS)

To measure requests per second (RPS), we ran the following script:

```
./run_wrk_namespace rps0k
./run_wrk_namespace rps1k
./run_wrk_namespace rps10k
./run_wrk_namespace rps100k
```

Transactions Per Second

To measure SSL/TLS transactions per second (TLS), we ran the following script:

```
./run_wrk_namespace tps
```

Throughput

To measure throughput, we ran the following script:

```
./run_wrk_namespace rps0k
./run_wrk_namespace rps1k
./run_wrk_namespace rps10k
./run_wrk_namespace rps100k
```

The only difference from the Throughput test and the RPS test is the larger file size of 100KB - 100MB. We also calculate the throughput from the transfers/sec output.

x Bytes per second = 8* bites per second

Running wrk in kernel

```
# cat go_namespace
#!/bin/bash

for MODE in kernel
do
  for i in `seq 1 28`
  do
    echo $MODE $i
    ssh sfocr740b "/root/Onload_Testing/NGINXPlus/start_nginx $MODE $i"
    sleep 20
    for TEST in tps rps0k rps1k rps10k rps100k thr1M thr10M thr100M
    do
      ./run_wrk_namespace $TEST
      sleep 210
  done
done
```

```
# cat run_wrk_namespace
#!/bin/bash
sysctl -w vm.nr_hugepages=2000
sysctl -w fs.file-max=8388608
sysctl -w fs.nr_open=8388608
ulimit -n 8388608

HOST=sfocr740b
TIME=180s
THREAD=1
CONNECTIONS=50
CPUS=`nproc`
let " CPUS = `nproc` - 1 "

INTERFACE=p2p1
IPADDR="192.168.102.35"

MODE=$1

case $MODE in
  tps)
    for i in `seq 0 $CPUS`
    do
      EF_MAX_ENDPOINTS=400000 \
      EF_UL_EPOLL=3 \
      EF_DYNAMIC_ACK_THRESH=4 \
      EF_TCP_TCONST_MSL=1 \
      EF_SCALABLE_FILTERS=any=rss:active:passive \
      EF_TCP_SHARED_LOCAL_PORTS_MAX=570000 CI_TP_LOG=7 \
      EF_EPOLL_SPIN=1 \
      EF_LOAD_ENV=1 \
      EF_USE_HUGE_PAGES=2 \
      EF_ACCEPTQ_MIN_BACKLOG=400 \
      EF_CLUSTER_RESTART=1 \
      EF_TCP_FIN_TIMEOUT=15 \
      EF_MAX_PACKETS=205000 \
      EF_TCP_SYNRECV_MAX=90000 \
      EF_TCP_BACKLOG_MAX=400 \
```

```
    EF_TCP_INITIAL_CWND=14600 \
    EF_TCP_SHARED_LOCAL_PORTS_REUSE_FAST=1 \
    EF_PIO=0 \
    EF_SLEEP_SPIN_USEC=50 LD_PRELOAD=libonload.so \
    EF_CLUSTER_SIZE=1 \
    EF_POLL_USEC=100000 \
    EF_UDP=0 \
    EF_SOCKET_CACHE_MAX=40000 \
    EF_FDTABLE_SIZE=8388608 \
    EF_TCP_SHARED_LOCAL_PORTS_PER_IP_MAX=570000 \
    EF_TCP_SHARED_LOCAL_PORTS_NO_FALLBACK=1 \
    EF_TAIL_DROP_PROBE=1 \
    EF_EPOLL_MT_SAFE=1 \
    EF_HIGH_THROUGHPUT_MODE=1 \
    EF_TCP_SHARED_LOCAL_PORTS=1000 \
    EF_LOG_VIA_IOCTL=1 \
    EF_TCP_SHARED_LOCAL_PORTS_PER_IP=1 \
    EF_NO_FAIL=1 \
    EF_CLUSTER_NAME=load \
    EF_MIN_FREE_PACKETS=50000 \
    EF_TX_PUSH=0 \
    EF_SCALABLE_FILTERS_ENABLE=1 \
    taskset -c $i ip netns exec net$i ./wrk -t $THREAD -c $CONNECTIONS \
            -d $TIME -H 'Connection: close' http://$IPADDR/0kb.bin \
            >> results/wrk_"$TEST"_"$INTERFACE"_"$i" &
  done

# for rps[0,1,10,100]K and thr[1,10.100]M, changed the file being called
# i.e.  http://$IPADDR/0kb.bin
rps0k)
  #Requests per second
  echo "============================="
  echo "REQUESTS per SEC"
  echo "-----------------------------"
  echo
  TEST="RPS0k"
  for i in `seq 0 $CPUS`
  do
    EF_MAX_ENDPOINTS=400000 \
    EF_UL_EPOLL=3 \
    EF_DYNAMIC_ACK_THRESH=4 \
    EF_TCP_TCONST_MSL=1 \
    EF_SCALABLE_FILTERS=any=rss:active:passive \
    EF_TCP_SHARED_LOCAL_PORTS_MAX=570000 CI_TP_LOG=7 \
    EF_EPOLL_SPIN=1 \
    EF_LOAD_ENV=1 \
    EF_USE_HUGE_PAGES=2 \
    EF_ACCEPTQ_MIN_BACKLOG=400 \
    EF_CLUSTER_RESTART=1 \
    EF_TCP_FIN_TIMEOUT=15 \
    EF_MAX_PACKETS=205000 \
    EF_TCP_SYNRECV_MAX=90000 \
    EF_TCP_BACKLOG_MAX=400 \
    EF_TCP_INITIAL_CWND=14600 \
    EF_TCP_SHARED_LOCAL_PORTS_REUSE_FAST=1 \
    EF_PIO=0 \
    EF_SLEEP_SPIN_USEC=50 LD_PRELOAD=libonload.so \
```

```
        EF_CLUSTER_SIZE=1 \
        EF_POLL_USEC=100000 \
        EF_UDP=0 \
        EF_SOCKET_CACHE_MAX=40000 \
        EF_FDTABLE_SIZE=8388608 \
        EF_TCP_SHARED_LOCAL_PORTS_PER_IP_MAX=570000 \
        EF_TCP_SHARED_LOCAL_PORTS_NO_FALLBACK=1 \
        EF_TAIL_DROP_PROBE=1 \
        EF_EPOLL_MT_SAFE=1 \
        EF_HIGH_THROUGHPUT_MODE=1 \
        EF_TCP_SHARED_LOCAL_PORTS=1000 \
        EF_LOG_VIA_IOCTL=1 \
        EF_TCP_SHARED_LOCAL_PORTS_PER_IP=1 \
        EF_NO_FAIL=1 \
        EF_CLUSTER_NAME=load \
        EF_MIN_FREE_PACKETS=50000 \
        EF_TX_PUSH=0 \
        EF_SCALABLE_FILTERS_ENABLE=1 \
        taskset -c $i ip netns exec net$i ./wrk -t $THREAD -c $CONNECTIONS \
                -d $TIME  http://$IPADDR/0kb.bin
    done
esac
```

5.4 Graphing the kernel benchmarking results

The results from each pass of `wrk` are now gathered and summed, so that they can be further analyzed. They are then transferred into an Excel spreadsheet, to create graphs from the data.

5.5 Cloud Onload benchmarking

The benchmarking is then repeated using Cloud Onload to accelerate NGINX Plus. To do so:

- create an Onload profile for NGINX, based on the supplied `latency-best` profile
- use Cloud Onload to accelerate `nginx` and `wrk`.

The nginx-balanced profile

```
# cat nginx_webserver_balanced.opf
#
# Tuning profile for nginx in reverse-proxy configuration with OpenOnload
# acceleration.
#
# User may supply the following environment variables:
#
#   NGINX_NUM_WORKERS        - the number of workers that nginx is
#                              configured to use. Overrides value
#                              automatically detected from nginx
#                              configuration
#

set -o pipefail

# For diagnostic output
module="nginx profile"

# Regular expressions to match nginx config directives
worker_processes_pattern="/(^|;)\s*worker_processes\s+(\w+)\s*;/"
include_pattern="/(^|;)\s*include\s+(\S+)\s*;/"

# Identify the config file that nginx would use
identify_config_file() {
  local file

  # Look for a -c option
  local state="IDLE"
  for option in "$@"
  do
    if [ "$state" = "MINUS_C" ]
    then
      file=$option
      state="FOUND"
    elif [ "$option" = "-c" ]
    then
      state="MINUS_C"
    fi
  done

  # Extract the compile-time default if config not specified on command
line
  if [ "$state" != "FOUND" ]
  then
```

```
    file=$($1 -h 2>&1 | perl -ne 'print $1 if
'"$worker_processes_pattern")
  fi

  [ -f "$file" ] && echo $file
}

# Recursively look in included config files for a setting of
worker_processes.
# NB If this quantity is set in more than one place then the wrong setting
might
# be found, but this would be invalid anyway and is rejected by Nginx.
read_config_file() {
  local setting
  local worker_values=$(perl -ne 'print "$2 " if
'"$worker_processes_pattern" $1)
  local include_values=$(perl -ne 'print "$2 " if '"$include_pattern" $1)

  # First look in included files
  for file in $include_values
  do
    local possible=$(read_config_file $file)
    if [ -n "$possible" ]
    then
      setting=$possible
    fi
  done

  # Then look in explicit settings at this level
  for workers in $worker_values
  do
    setting=$workers
  done
  echo $setting
}

# Method to parse configuration files directly
try_config_files() {
  local config_file=$(identify_config_file "$@")
  [ -n "$config_file" ] && read_config_file $config_file
}

# Method to parse configuration via nginx, if supported
try_nginx_minus_t() {
  "$@" -T | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Method to parse configuration via tengine, if supported
try_tengine_minus_d() {
  "$@" -d | perl -ne 'print "$2" if '"$worker_processes_pattern"
}

# Determine the number of workers nginx will use
determine_worker_processes() {
  # Prefer nginx's own parser, if available, for robustness
  local workers=$(try_nginx_minus_t "$@" || try_tengine_minus_d "$@" ||
try_config_files "$@")
```

```
  if [ "$workers" = "auto" ]
  then
    # Default to the number of process cores
    workers=$(nproc)
  fi
  echo $workers
}

# Define the number of workers
num_workers=${NGINX_NUM_WORKERS:-$(determine_worker_processes "$@")}
if ! [ -n "$num_workers" ]; then
  fail "ERROR: Environment variable NGINX_NUM_WORKERS is not set and
worker count cannot be determined from nginx configuration"
fi
log "$module: configuring for $num_workers workers"

# nginx uses epoll within one process only
onload_set EF_EPOLL_MT_SAFE 1

# Enable clustering to spread connections over workers.
onload_set EF_CLUSTER_SIZE "$num_workers"
onload_set EF_CLUSTER_NAME webs
onload_set EF_CLUSTER_RESTART 1
onload_set EF_CLUSTER_HOT_RESTART 1

# Enable spinning and sleep-spin mode.
onload_set EF_POLL_USEC 1000000
onload_set EF_SLEEP_SPIN_USEC 50

onload_import throughput
onload_import wan-traffic

# In case invocation tries to send signal to existing instance of nginx
# omit stack checking.
if echo "$@" | perl -n -e 'if(/\s-s/) {exit 1}'; then
  # In case of cold restart make sure previous instance (of the same name)
has
  # ceased to exist and in case references to onload stacks are still
being
  # released - wait.

  ITER=0
  while onload_stackdump --nopids stacks | grep "\s${EF_CLUSTER_NAME}-c"
>/dev/null; do
    if (( $ITER % 20 == 19 )); then
      echo Onload stacks of name ${EF_CLUSTER_NAME}-c## still present. >&2
      echo Verify that previous instance of Nginx has been killed. >&2
      onload_stackdump --nopids stacks >&2
      if (( $ITER > 50 )); then
        exit 16
      fi
    fi
    ITER=$(( $ITER + 1 ))
    sleep 0.2;
  done
fi
```

The nginx-performance profile

The nginx-performance profile is almost identical to the nginx-balanced profile:

```
# diff nginx_webserver_balanced.opf nginx_webserver_spinning.opf
121d120
< onload_set EF_SLEEP_SPIN_USEC 50
```

Use Cloud Onload to accelerate the software

Repeat the testing using Cloud Onload to accelerate NGINX Plus. Precede each command with:

```
onload --profile=nginx-balanced
```

or:

```
onload --profile=nginx-performance
```

Accelerating NGINX Plus

To accelerate NGINX Plus, edit the start script:

- uncomment the line that uses Cloud Onload
- comment out the line that runs unaccelerated NGINX Plus.

```
# cat start_nginx
#!/bin/bash

#PROFILE="nginx_webserver_balanced.opf"
PROFILE="nginx_webserver_performance.opf"

killall -9 nginx
NPROCS=$2
sysctl -w vm.nr_hugepages=10000;
sysctl -w fs.file-max=8388608;
sysctl -w fs.nr_open=8388608;
ulimit -n 8388608;

onload -p $PROFILE nginx \
  -c /root/Onload_Testing/NGINXPlus/nginx_webserver/nginx-webserver_"$NPROCS".conf

echo "PID of nginx"
pidof nginx
echo "Stackdump"
onload_stackdump
echo "Stackdump clusters size"
onload_stackdump clusters | grep size
```

6 Benchmark results

This chapter presents the benchmark results that are achieved. See:

6.1 Results

Transactions per second (TPS) at 25Gb/s

The following command line was used:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s -H 'Connection: close' \
  https://192.168.105.35/0kb.bin
```

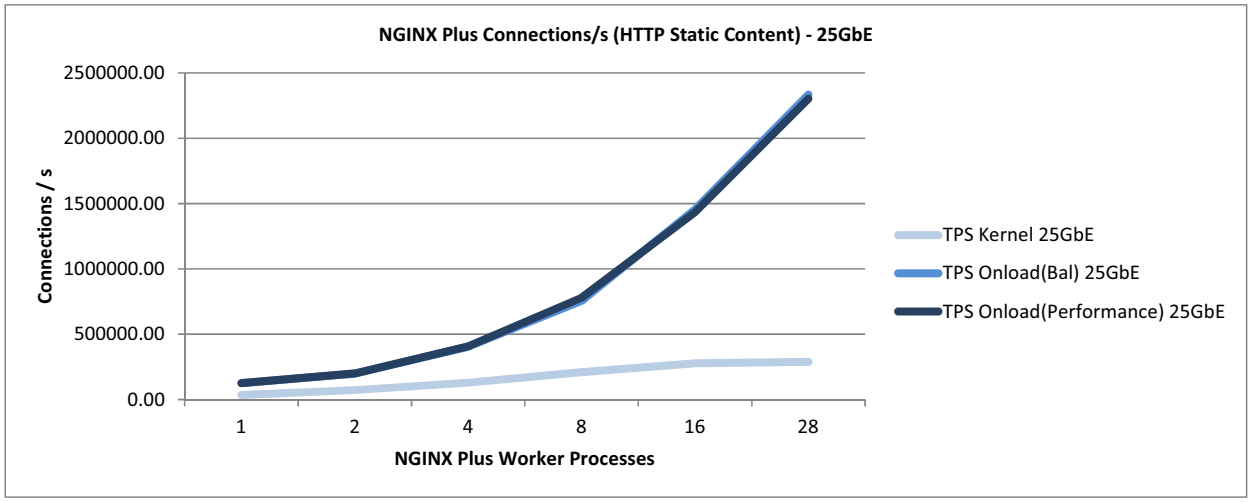

Figure 5: NGINX Plus transactions per second at 25Gb/s

Table 2 below shows the results that were used to plot the graph in Figure 5 above.

Table 2: Transactions per second at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	35890.70	126007.00	125273.00	251.09%	249.04%
2	73226.10	198366.00	201056.00	170.90%	174.57%
4	128555.00	404713.00	407412.00	214.82%	216.92%
8	209513.00	755081.00	779808.00	260.40%	272.20%
16	278366.00	1454320.00	1434150.00	422.45%	415.20%
28	288632.00	2334780.00	2302380.00	708.91%	697.69%

Requests per second (RPS) at 25Gb/s

The following command lines were used:

- RPS using a 0KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/0kb.bin
```

- RPS using a 1KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/1kb.bin
```

- RPS using a 10KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/10kb.bin
```

- RPS using a 100KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100kb.bin
```

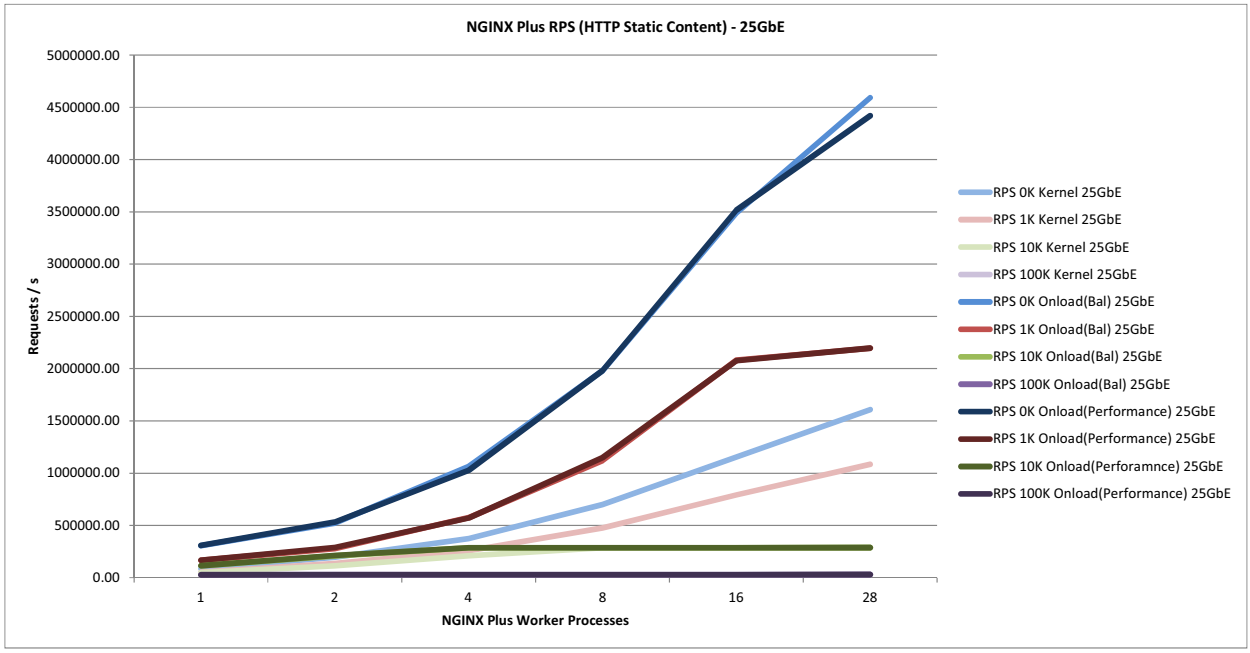

Figure 6: NGINX Plus requests per second at 25Gb/s

Table 3 to Table 6 inclusive below show the results that were used to plot the graph in Figure 6 above.

Table 3: Requests per second for 0KB at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	86004.50	306892.00	308000.00	256.83%	258.12%
2	194653.00	521760.00	531806.00	168.05%	173.21%
4	373281.00	1064170.00	1027180.00	185.09%	175.18%
8	698718.00	1979270.00	1981920.00	183.27%	183.65%
16	1153630.00	3488740.00	3516600.00	202.41%	204.83%
28	1608500.00	4592430.00	4420920.00	185.51%	174.85%

Table 4: Requests per second for 1KB at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	52367.50	165584.00	167565.00	216.20%	219.98%
2	136150.00	277197.00	288077.00	103.60%	111.59%
4	258389.00	573284.00	570791.00	121.87%	120.90%
8	474954.00	1120850.00	1149680.00	135.99%	142.06%
16	791130.00	2080510.00	2075420.00	162.98%	162.34%
28	1084600.00	2196410.00	2196730.00	102.51%	102.54%

Table 5: Requests per second for 10KB at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	48694.90	113308.00	114581.00	132.69%	135.30%
2	112563.00	208214.00	211525.00	84.98%	87.92%
4	209280.00	286513.00	286615.00	36.90%	36.95%
8	287472.00	286772.00	286721.00	-0.24%	-0.26%
16	287469.00	286660.00	286597.00	-0.28%	-0.30%
28	287497.00	287355.00	287409.00	-0.05%	-0.03%

Table 6: Requests per second for 100KB at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	27083.30	27266.10	26967.30	0.67%	-0.43%
2	29723.50	29529.20	29521.00	-0.65%	-0.68%
4	29584.00	29518.40	29505.00	-0.22%	-0.27%
8	29577.60	29520.40	29505.00	-0.19%	-0.25%
16	29574.20	29515.80	29492.00	-0.20%	-0.28%
28	29571.40	29559.00	29567.60	-0.04%	-0.01%

Throughput at 25Gb/s

The following command lines were used:

- Throughput using a 100K:B file

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100kb.bin
```

- Throughput using a 1MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/1Mb.bin
```

- Throughput using a 10MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/10Mb.bin
```

- Throughput using a 100MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100Mb.bin
```

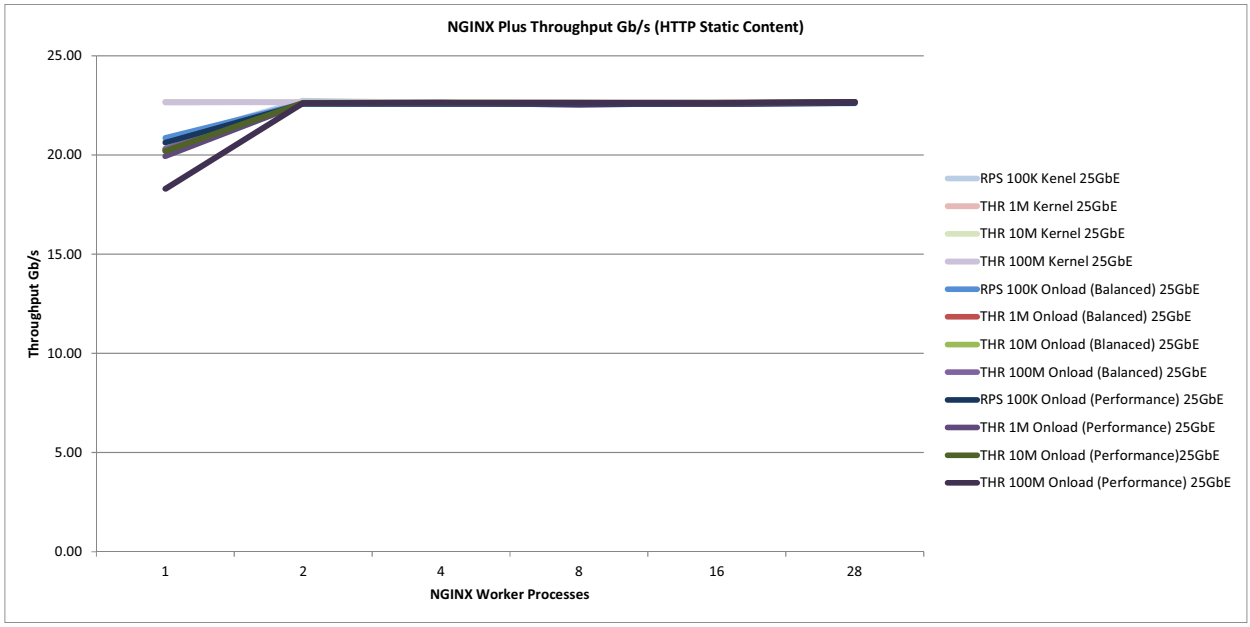

Figure 7: NGINX Plus throughput at 25Gb/s

Table 7 to Table 10 inclusive below show the results that were used to plot the graph in Figure 7 above.

Table 7: Throughput for 100K at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	20.72	20.86	20.63	0.67%	-0.43%
2	22.74	22.59	22.58	-0.67%	-0.69%
4	22.63	22.58	22.57	-0.24%	-0.28%
8	22.63	22.58	22.57	-0.21%	-0.26%
16	22.62	22.58	22.56	-0.21%	-0.29%
28	22.62	22.61	22.62	-0.05%	-0.02%

Table 8: Throughput for 1M at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	22.68	20.26	19.94	-10.69%	-12.10%
2	22.65	22.62	22.61	-0.16%	-0.18%
4	22.64	22.63	22.62	-0.06%	-0.10%
8	22.65	22.63	22.61	-0.07%	-0.15%
16	22.64	22.62	22.58	-0.10%	-0.31%
28	22.64	22.65	22.66	0.04%	0.08%

Table 9: Throughput for 10M at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	22.66	20.31	20.20	-10.34%	-10.85%
2	22.66	22.62	22.62	-0.16%	-0.17%
4	22.65	22.64	22.63	-0.04%	-0.09%
8	22.65	22.64	22.62	-0.07%	-0.15%
16	22.65	22.63	22.61	-0.10%	-0.17%
28	22.65	22.67	22.67	0.06%	0.07%

Table 10: Throughput for 100M at 25Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	22.66	20.31	18.29	-10.36%	-19.25%
2	22.66	22.62	22.62	-0.21%	-0.20%
4	22.66	22.64	22.63	-0.07%	-0.12%
8	22.66	22.52	22.63	-0.63%	-0.13%
16	22.66	22.63	22.61	-0.12%	-0.21%
28	22.66	22.67	22.67	0.02%	0.05%

Transactions per second (TPS) at 100Gb/s

The following command line was used:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s -H 'Connection: close' \
  https://192.168.105.35/0kb.bin
```

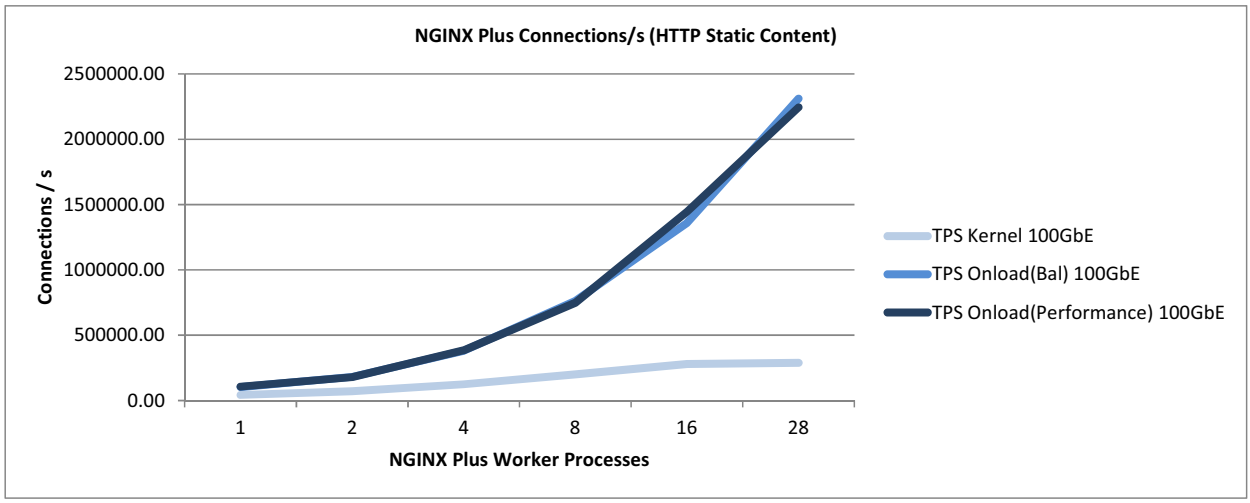

Figure 8: NGINX Plus transactions per second

Table 11 below shows the results that were used to plot the graph in Figure 8 above.

Table 11: Transactions per second at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	42836.30	103569.00	106688.00	141.78%	149.06%
2	70545.40	180855.00	180235.00	156.37%	155.49%
4	125168.00	381044.00	385242.00	204.43%	207.78%
8	200622.00	763533.00	749029.00	280.58%	273.35%
16	280856.00	1359440.00	1444500.00	384.03%	414.32%
28	289087.00	2310860.00	2244150.00	699.36%	676.29%

Requests per second (RPS) at 100Gb/s

The following command lines were used:

- RPS using a 0KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/0kb.bin
```

- RPS using a 1KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/1kb.bin
```

- RPS using a 10KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/10kb.bin
```

- RPS using a 100KB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100kb.bin
```

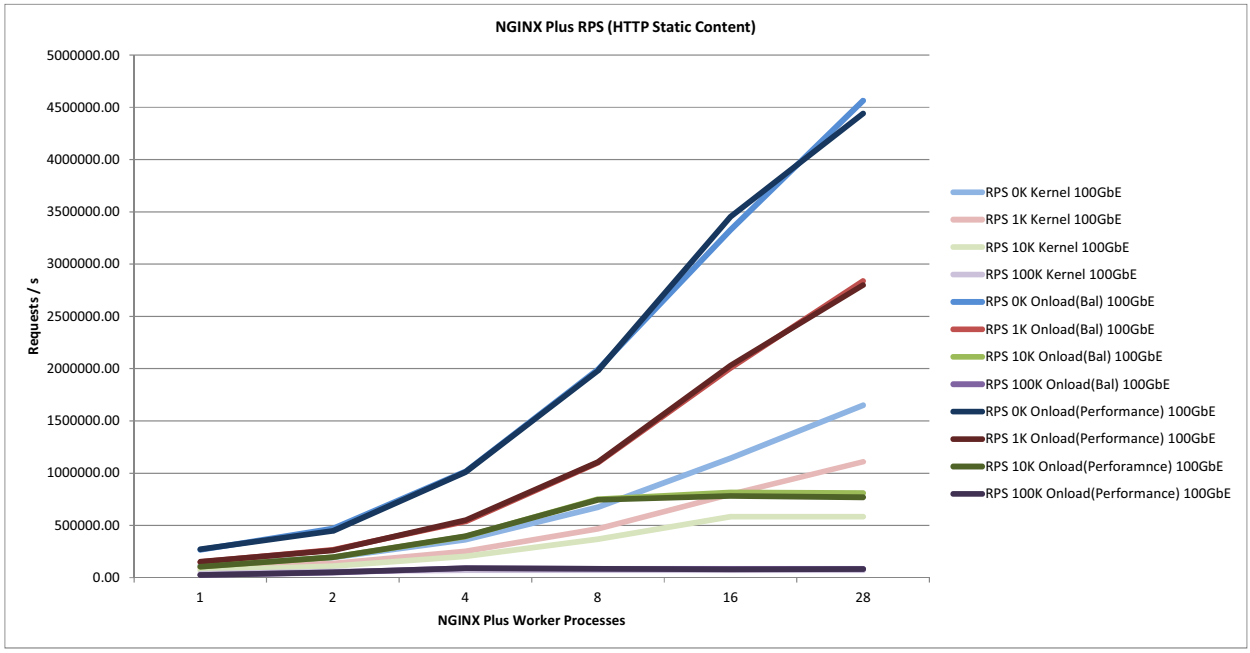

Figure 9: NGINX Plus requests per second at 100Gb/s

Table 12 to Table 15 inclusive below show the results that were used to plot the graph in Figure 9 above.

Table 12: Requests per second for 0KB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	119808.00	265043.00	271094.00	121.22%	126.27%
2	191987.00	470043.00	446627.00	144.83%	132.63%
4	363522.00	1014400.00	1009430.00	179.05%	177.68%
8	672968.00	1993100.00	1980210.00	196.17%	194.25%
16	1143940.00	3326530.00	3453830.00	190.80%	201.92%
28	1650560.00	4564640.00	4441240.00	176.55%	169.07%

Table 13: Requests per second for 1KB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	89494.20	148669.00	152095.00	66.12%	69.95%
2	135374.00	266472.00	260857.00	96.84%	92.69%
4	252477.00	537498.00	549391.00	112.89%	117.60%
8	466750.00	1099630.00	1105430.00	135.59%	136.84%
16	796606.00	2005890.00	2030280.00	151.80%	154.87%
28	1108940.00	2838660.00	2799480.00	155.98%	152.45%

Table 14: Requests per second for 10KB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	68673.70	105066.00	105900.00	52.99%	54.21%
2	109814.00	193849.00	194762.00	76.52%	77.36%
4	205417.00	392281.00	396857.00	90.97%	93.20%
8	368371.00	752507.00	744910.00	104.28%	102.22%
16	583840.00	815065.00	781134.00	39.60%	33.79%
28	583526.00	809674.00	769008.00	38.76%	31.79%

Table 15: Requests per second for 100KB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	36765.10	25497.70	25453.70	-30.65%	-30.77%
2	60136.30	49940.20	50734.70	-16.95%	-15.63%
4	71832.30	91516.00	91158.30	27.40%	26.90%
8	71611.10	83745.50	84585.30	16.94%	18.12%
16	70994.80	83947.20	80096.20	18.24%	12.82%
28	70588.20	84595.90	82335.20	19.84%	16.64%

Throughput at 100Gb/s

The following command lines were used:

- Throughput using a 100K:B file

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100kb.bin
```

- Throughput using a 1MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/1Mb.bin
```

- Throughput using a 10MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/10Mb.bin
```

- Throughput using a 100MB file:

```
# taskset -c [0-31] ./wrk -t 1 -c 50 -d 180s \
  http://192.168.105.35/100Mb.bin
```

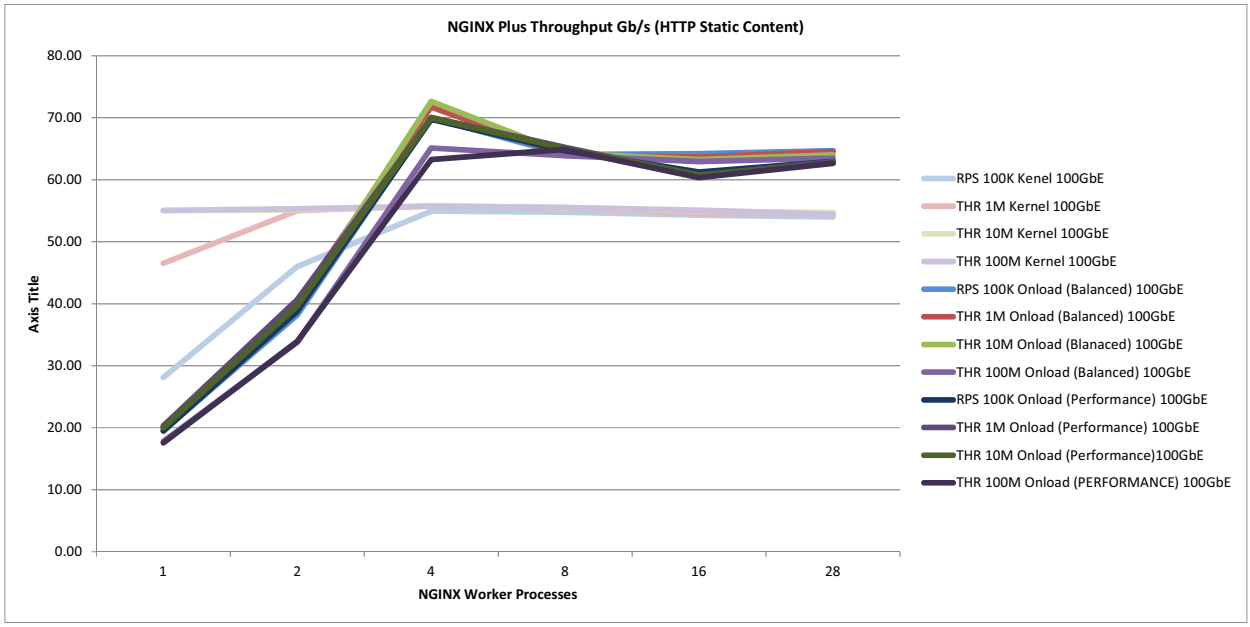

Figure 10: NGINX Plus throughput at 100Gb/s

Table 16 to Table 19 inclusive below show the results that were used to plot the graph in Figure 10 above.

Table 16: Throughput for 100KB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	28.12	19.50	19.47	-30.65%	-30.77%
2	46.00	38.20	38.81	-16.95%	-15.63%
4	54.95	70.00	69.73	27.40%	26.90%
8	54.78	64.06	64.70	16.94%	18.11%
16	54.31	64.21	61.27	18.24%	12.82%
28	54.00	64.71	62.98	19.84%	16.64%

Table 17: Throughput for 1MB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	46.51	20.16	20.31	-56.65%	-56.33%
2	55.02	40.26	40.65	-26.82%	-26.11%
4	55.81	71.75	70.05	28.56%	25.51%
8	55.46	63.92	65.21	15.25%	17.57%
16	54.54	63.70	60.71	16.80%	11.31%
28	54.60	64.55	62.92	18.23%	15.23%

Table 18: Throughput for 10MB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	55.05	19.75	20.00	-64.12%	-63.67%
2	55.14	39.49	39.67	-28.39%	-28.06%
4	55.75	72.64	69.97	30.29%	25.51%
8	55.51	64.28	65.02	15.80%	17.13%
16	54.95	63.25	60.79	15.10%	10.61%
28	54.72	64.00	62.76	16.97%	14.69%

Table 19: Throughput for 100MB at 100Gb/s

Worker processes	Kernel	Onload balanced	Onload performance	Onload balanced gain	Onload performance gain
1	55.04	17.78	17.57	-67.71%	-68.08%
2	55.30	33.80	33.94	-38.87%	-38.63%
4	55.71	65.11	63.26	16.88%	13.56%
8	55.53	63.86	64.90	14.99%	16.87%
16	55.08	62.94	60.35	14.28%	9.58%
28	54.41	63.49	62.65	16.70%	15.15%

6.2 Analysis

Transactions per second

For 100GbE and 25GbE at 16-28× worker processes we see a flattening out for kernel processes. While Cloud Onload shows a significant linear scaling to at least 676% gain, at both speeds, and with both balanced and performance profiles.

Both 100GbE and 25GbE show that for the current configuration (1× NGINX server plus 1× wrk generator server), when the NGINX server uses Cloud Onload we cannot saturate it with enough wrk connections.

Requests per second

Large HTTP requests (such as the 10KB and 100 KB sizes in the test) are fragmented and take longer to process. As a result, the lines in the graph for larger requests have flatter slopes.

- For 100KB files at 100GbE, Cloud Onload delivers its greatest gains of over 26% with 4× NGINX worker processes. Gains then flatten out at 12% or above with more than 8× worker processes.
- For 100KB files at 25GbE, there is no difference between kernel and Cloud Onload.
- For 10KB files at 100GbE, Cloud Onload delivers gains increasing from 52% to 102% as NGINX worker processes increase from 1× to 8×, at which point Cloud Onload flattens out. The kernel worker processes do not saturate until there are 16× worker processes, after which Cloud Onload still shows gains of greater that 31%.

- For 10KB files at 25GbE, Cloud Onload delivers gains of more than132% for a single worker process, and gains of greater than 36% with up to 4× NGINX worker processes. With 8× NGINX worker processes or more, the processes are saturated, and the kernel and Cloud Onload show similar results.
- For 0KB to 1KB files at 100GbE, we see peak gains of greater than 201% and 154% respectively. These peaks occur between 8× and 16× worker processes for 0K files, and between 16× to 28× worker processes for 1K files. Note that we do not see saturation of the kernel or Cloud Onload worker processes.
- For 0KB to 1KB files at 25GbE, we saturate at 16× worker processes for 1K files. Peak gains at 16× worker processes are greater than 202% and 162% respectively.

Throughput

For extra large files (100KB to 100MB) at 100GbE, Cloud Onload gets peak performance gains with 4× worker processes, delivering 30.29% gain for 10MB files and 16.88% gain for 100MB files. The worker processes are saturated around 54 Gbps for the kernel and 64 Gbps for Cloud Onload.

For extra large files (100KB to 100MB) at 25GbE, the NGINX worker processes are saturated by 2× worker processes for both Cloud Onload and the kernel.