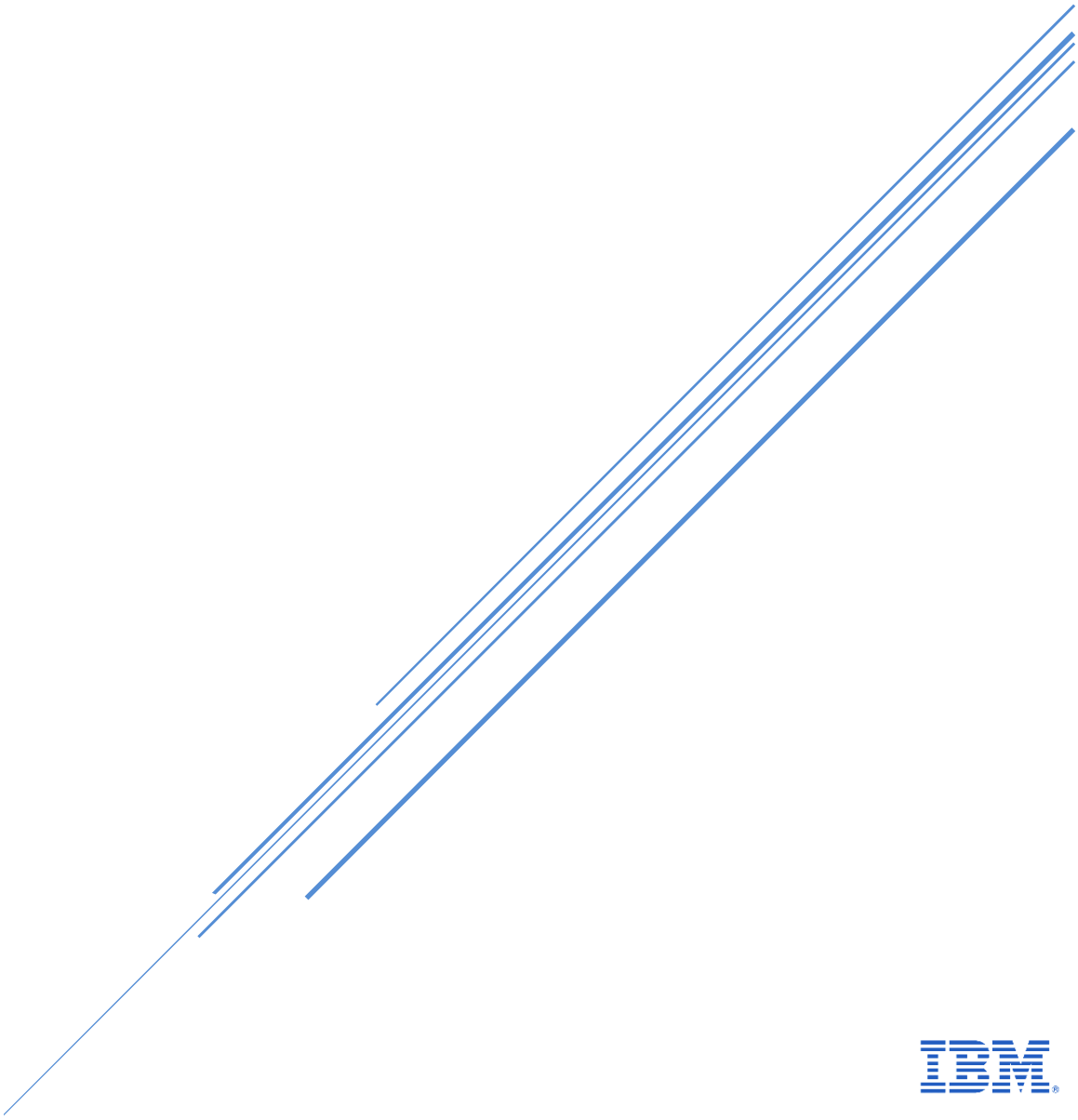
Industry Standard Performance Evaluation Kit

IBM OpenPOWER Performance Enablement

How to run the sockperf benchmarking tool



sockperf

sockperf is a network benchmarking utility over socket API that was designed for testing performance (latency and throughput) of high-performance systems (it is also good for testing performance of regular networking systems as well). It covers most of the socket API calls and options.

Follow the directions provided in this Github branch for obtaining and building sockperf.

For additional information visit their website at http://code.google.com/p/sockperf.

Running sockperf

An instance of sockperf must be running on both a client and a server.

```
Usage: sockperf <subcommand> [options] [args]
Type: 'sockperf <subcommand> --help' for help on a specific subcommand.
Type: 'sockperf --version' to see the program version number.
```

Available subcommands:

```
   help (h ,?)          Display list of supported commands.
   under-load (ul)      Run sockperf client for latency under load test.
   ping-pong (pp)       Run sockperf client for latency test in ping pong mode.
   playback (pb)        Run sockperf client for latency test using playback of
predefined traffic, based on timeline and message size.
   throughput (tp)      Run sockperf client for one way throughput test.
   server (sr)          Run sockperf as a server.
```

1. Starting sockperf as a server on HOST A (10.0.10.101):

```
[HOST A] # sockperf server --ip 10.0.0.101
sockperf: == version #2.8-1.git9fbe737a4d29 ==
sockperf: [SERVER] listen on:
[ 0] IP = 10.0.0.101      PORT = 11111 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: [tid 18975] using recvfrom() to block on socket(s)
^Csockperf: Test end (interrupted by user)
sockperf: No messages were received on the server.
sockperf: cleanupAfterLoop() exit
[root@hadoop1 bengibbs]# ./sockperf server --tcp --ip 10.0.0.101
sockperf: == version #2.8-1.git9fbe737a4d29 ==
sockperf: [SERVER] listen on:
[ 0] IP = 10.0.0.101      PORT = 11111 # TCP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: [tid 18976] using recvfrom() to block on socket(s)
```

2. Running sockperf on client using ping pong (pp) to HOST A at 10.0.0.101 and running for 30 seconds.

```
[HOST B] # sockperf pp --tcp --ip 10.0.0.101 -t 30
sockperf: == version #2.8-1.git9fbe737a4d29 ==
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on socket(s)

[ 0] IP = 10.0.0.101      PORT = 11111 # TCP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: Starting test...
sockperf: Test end (interrupted by timer)
sockperf: Test ended
sockperf: [Total Run] RunTime=30.100 sec; SentMessages=736026;
ReceivedMessages=736025
sockperf: ========= Printing statistics for Server No: 0
```

```
sockperf: [Valid Duration] RunTime=30.000 sec; SentMessages=733574;
ReceivedMessages=733574
sockperf: ====> avg-lat= 20.421 (std-dev=0.593)
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-order
messages = 0
sockperf: Summary: Latency is 20.421 usec
sockperf: Total 733574 observations; each percentile contains 7335.74
observations
sockperf: ---> <MAX> observation =  279.592
sockperf: ---> percentile 99.999 =   41.714
sockperf: ---> percentile 99.990 =   25.824
sockperf: ---> percentile 99.900 =   23.390
sockperf: ---> percentile 99.000 =   22.300
sockperf: ---> percentile 90.000 =   20.951
sockperf: ---> percentile 75.000 =   20.500
sockperf: ---> percentile 50.000 =   20.308
sockperf: ---> percentile 25.000 =   20.166
sockperf: ---> <MIN> observation =   14.439
```

sock_stress.sh

A test script is provided to ease the testing effort. The name of the script is called *sock_stress.sh*. This script is ran on both the server and the client. The script allows you to run multiple instances of sockperf on the client. Each instance will be NUMA bound to the primary thread of each core before wrapping back to the secondary thread of each core and so on.

NOTE: If you are having trouble with the sockperf client communicating with the sockperf server, chances are the Linux firewall is blocking the communication ports. To open up the ports on both the client and the server you will need to run the following command:

```
# iptables -F
```

Running the sock_stress.sh script

1. On the server, run the sockperf benchmark with the default settings, using a buffer size of 1MB and 8 listening ports.

```
[HOST A] # sock_stress.sh -b 1048576 -s -n 8
```

2. On the client, run the sockperf benchmark on 8 threads, using a buffer size of 1MB, a message size of 512 bytes for 30 seconds.

```
[HOST B] # sock_stress.sh -b 1048576 -m 512 -n 8 -t 30
Collecting data for message size of 512
Processing /tmp/sockperf_0
Processing /tmp/sockperf_16
Processing /tmp/sockperf_24
Processing /tmp/sockperf_32
Processing /tmp/sockperf_40
Processing /tmp/sockperf_48
Processing /tmp/sockperf_56
Processing /tmp/sockperf_8
```

3. A results file named *sockperf.results* is produced on both the server and the client. This is a sample output using the example shown above for the client:

```
# cat sockperf.results
SOCKPERF TEST RESULTS
=====================
Hostname:              hostB
Architecture:          ppc64le
Byte Order:            Little Endian
CPU(s):                160
On-line CPU(s) list:   0-159
Thread(s) per core:    8
Core(s) per socket:    10
Socket(s):             2
NUMA node(s):          2
Model:                 JVASB1008550 B01
L1d cache:             64K
L1i cache:             32K
L2 cache:              512K
L3 cache:              8192K
NUMA node0 CPU(s):     80-159
NUMA node8 CPU(s):     0-79
=========================================
= netstat -i output BEFORE
=========================================
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enP1p1s0  9000 284863332      0      0 0      9411499933      0      0      0
BMRU
enP1p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP2p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP2p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP4p10s  1500 11160307      0   6672 0        43254      0      0      0 BMRU
enP4p10s  1500        0      0      0 0            0      0      0      0 BMU
lo       65536       69      0      0 0           69      0      0      0 LRU
=========================================
= netstat -i output AFTER
=========================================
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enP1p1s0  9000 289379055      0      0 0      9428149265      0      0      0
BMRU
enP1p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP2p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP2p1s0  1500        0      0      0 0            0      0      0      0 BMU
enP4p10s  1500 11161028      0   6673 0        43256      0      0      0 BMRU
enP4p10s  1500        0      0      0 0            0      0      0      0 BMU
lo       65536       69      0      0 0           69      0      0      0 LRU
=========================================
= Results for each thread
=========================================
Messages   Size    Secs Msgs/s MB/sec   Mb/s
24262579    512  30.10 806068  393.59 3148.70
48466496    512  30.10 1610185 786.22 6289.78
24251847    512  30.10 805710  393.41 3147.30
24252010    512  30.10 805719  393.42 3147.34
48497904    512  30.10 1611232 786.73 6293.88
24262392    512  30.10 806061  393.58 3148.68
48500616    512  30.10 1611325 786.78 6294.24
48469379    512  30.10 1610281 786.27 6290.16
```