SAND2019-13444

SANDIA REPORT

Printed

Sandia National Laboratories

RoCE: Promising Technology for Ethernet as a High Performance Networking Fabric

Joseph P. Kenny and Craig D. Ulmer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

Abstract

Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) has the potential to provide performance that rivals traditional high performance fabrics. If this potential proves out, significant impacts on system procurement decisions could follow. This work provides a series of small scale performance results which are used to compare and contrast the performance of RoCE-enabled Ethernet with TCP-based Ethernet and an HPC network. Additionally, a discussion of the maturity of RoCE firmware/software stacks and documentation is provided along with useful approaches for probing performance. A detailed description of two experimental setups known to have good RoCE performance is given, including step-by-step configuration and the exact hardware and software revisions employed. At small scales, RoCE is found to have significant performance advantages over "out-of-the-box" TCP protocols and is competitive with state-of-the-art high performance networks. Further examination of RoCE using a wider array of benchmarks and at greater scale is warranted.

Acknowledgment

We are thankful for the support and assistance of several colleagues. Gregory Faussette and Kurt Rago from Mellanox Technologies arranged the loan of a Spectrum SN2700 switch for this work and participated in valuable technical discussions. Jerry Friesen provided project support and management, and handled procurement activities. Jerry Friesen, Gavin Baker and Samuel Knight participated in technical discussions and assisted in computer system management.

Contents

List of Figures

List of Tables

1. Introduction

Our facility recently installed a 288 node Linux cluster, named Carnac, to support growing network emulation and cybersecurity workloads. [5] Such work typically involves standing up large numbers of virtual machines on the system. Thus, this cluster was procured with a director-class 100Gb/s Ethernet switch (Arista DCS-7512N) in order to keep virtual machine configuration easy and ensure that the switch data plane would have sufficient resources for the large number of virtual hosts. Carnac was added to an existing environment which uses an Infiniband interconnect to support more traditional high performance computing (HPC) workloads, depending largely upon message passing (MPI). A significant increase in flexibility and economy of scale would ensue if both user communities could be supported by a single switching fabric. If Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) could adequately support MPI workloads on Ethernet fabrics, we could dynamically provision Carnac based on user demand as well as consider procuring future clusters with a single Ethernet fabric to support our entire range of workloads. [8, 9, 3]

In this report we describe the work which we have performed to date evaluating RoCE-enabled Ethernet as an HPC interconnect. While our experience is that small to mid-scale Infiniband/Omni-Path clusters are more or less plug-and-play from a performance standpoint, there is significant overhead in understanding how to properly configure networks for RoCE. Additionally, and quite impactful in our investigation, RoCE technologies are evolving relatively rapidly and struggling to reach a state of maturity. Our current conclusion is that at small scale RoCE can perform very well, but the path to reaching that conclusion was fairly arduous. We hope to provide enough detail that this work can serve as a primer to assist further research on RoCE. In the following sections we provide a simplified overview of RoCE as it pertains to HPC workloads, detail our initial struggles with RoCE along with some helpful strategies for probing performance, and provide limited small-scale performance results for a well-specified set of hardware/software components demonstrated to provide good RoCE performance.

2. RDMA over Converged Ethernet

First, it is important to remind ourselves that Ethernet development is driven by the needs of large commercial data centers, not the needs of HPC. [6] It is, in fact, the desire to eliminate Fibre Channel storage networks and use a single "converged Ethernet" network within data centers that has largely driven the development of the technology which enables RoCE. The IEEE 802.1 Data Center Bridging Task Group standardized a number of technologies, typically referred to collectively as Data Center Bridging (DCB), supporting converged Ethernet. Protocols like Fibre Channel over Ethernet are sensitive to packet loss, so the best-effort or lossy nature of Ethernet is a serious obstacle for converged data center Ethernet. Priority Flow Control (PFC) was developed as an extension to earlier global flow control capabilities, allowing the fabric to pause flows belonging to selected priority levels in response to congestion. The ability to pause flows before resources are overwhelmed and packets need to be dropped allows the fabric to appear lossless under most operating conditions, though the credit-based flow control commonly found in HPC networks is more robust under extreme conditions. Combining PFC with quality of service (QoS) capabilities, storage traffic can be given a dedicated, essentially lossless share of the converged network.

Similar to Fibre Channel, the RDMA protocols which allow high-performance MPI implementations to bypass the kernel for messaging operations also assume a lossless fabric. With flow control providing a "lossless" Ethernet, RDMA operations can be performed by encapsulating Infiniband (IB) packets in Ethernet frames, and the RoCE standards specify how to do this. [4] There are v1 and v2 RoCE standards. Unlike RoCE v1, RoCE v2 is routable (can work across multiple subnets). Additionally, some switches may have poor performance for RoCE v1 – so it seems prudent to stick with RoCE v2 although in most small HPC clusters there is likely no difference in performance.

3. Experimental Setups

The benchmark results reported in this work were obtained using the experimental setups described in Table 3-1.

Carnac	
CPU	2x Intel E5-2683 v4
OS/Kernel	CentOS Linux release 7.6.1810 / Linux 3.10.0-957.el7.x86_64
NIC	Mellanox ConnectX-5 MT27800, hw_ver: 0x0, board_id: MT_0000000011
NIC Drivers/Firmware	MLNX_OFED_LINUX-4.6-1.0.1.1, fw_ver: 16.24.1000
Switch	Arista DCS-7512N 100Gb/s Ethernet, HW Version: 14.00
Switch OS	Arista EOS, Software image version 4.17.6M
Carnac/mlx	
CPU	2x Intel E5-2683 v4
OS/Kernel	CentOS Linux release 7.6.1810 / Linux 3.10.0-957.el7.x86_64
NIC	Mellanox ConnectX-5 MT27800, hw_ver: 0x0, board_id: MT_0000000011
NIC Drivers/Firmware	MLNX_OFED_LINUX-4.6-1.0.1.1, fw_ver: 16.24.1000
Switch	Mellanox SN2700 100Gb/s Ethernet, HW Rev. B2
Switch OS	Mellanox Onyx, version 3.8.1304
Serrano	
CPU	2x Intel E5-2695 v4
OS/Kernel	Red Hat Enterprise Linux Server release 7.7 / Linux 3.10.0-1062.1.1.1chaos.ch6.x86_64
NIC	Intel Omni-Path HFI 100, hw_ver: 0x11
NIC Drivers/Firmware	fw_ver: 1.26.1
Switch	Intel Edge Switch 100 100Gb/s Omni-Path

Table 3-1. Experimental setups.

4. RoCE Configuration

In the end, it turned out to be quite straightforward to configure a small Ethernet network for good RoCE performance. However, arriving at that configuration took considerable time and effort. Documentation for RoCE is quite limited. Switch user manuals specify how to turn features on and off, but do not go into any detail on the theory of the features or how they should be employed in concert to reach a certain result. Many internet articles provide step by step instructions, but these are very specific as to hardware, software, configuration utilities, and network architecture; such documentation goes stale quickly and over time the volume of articles confuses efforts to determine the ideal configuration approach. Additionally, the churn of Ethernet standards, networking hardware and software, and even Linux kernels provides so many moving targets that the ecosystem lacks maturity and stability.

We settled on what eventually proved to be an effective RoCE configuration for the Carnac cluster relatively quickly. While initial point-to-point MPI tests were very encouraging, we quickly ran into a roadblock running High Performance Linpack (HPL) [2] at scales near full machine size (288 nodes) where HPL would fall off a performance cliff and essentially cease to make progress. It seemed likely that flow control was not working adequately and that congestion at this scale was causing packet loss. While attempting to analyze the poor HPL performance, it became apparent that the MPI implementations which HPL was running on top of were often obscuring exactly how the network stack was being utilized. Depending on the MPI implementation used, performance variation was high and more importantly transports which failed would be quietly swapped out at runtime for the TCP transport. At times this fallback to TCP was not even indicated by MPI debugging output and was only recognized through examination of hardware performance counters. Discouraged by this setback, we decided to remove the MPI wild card and develop a simpler benchmark as an application proxy.

From user space, RoCE-capable hardware appears much like Infiniband hardware, including the availability of an IBVerbs interface which can be used to perform RDMA operations. Thus, most of the typical Infiniband tools (e.g. `ibstatus`) are installed with Mellanox's Open Fabrics Enterprise Distribution (OFED) package. While we could have written a benchmark directly to the IBVerbs interface, the availability of `ib_write_bw` allowed the rapid development of a Python script which can setup patterns of RDMA traffic and report observed bandwidths. As many-to-one incast can rapidly cause congestion in network links, we used incast patterns created with this script as a simple proxy for high bandwidth HPC applications. The congestion created by this incast benchmark can be tuned by varying the number of sources. This benchmark was used for subsequent debugging and testing, and results from it will be presented in Section 5 and Section 6.

On the Carnac Arista switch the incast benchmark was able to easily create congestion and large numbers of dropped packets using only a handful of nodes. This was exhibited in our testing as a significant drop-off in aggregate throughput for larger message sizes as the number of source nodes was increased with our incast benchmark. After exhausting our configuration options with the Arista switch, we obtained a loaned Spectrum SN2700 switch from Mellanox for further testing. Initial testing with the Mellanox switch showed performance problems very similar to those observed on the Arista switch. At this point we were mostly convinced that Ethernet flow control just isn't responsive enough to handle heavy congestion without dropping packets and causing severe RoCE performance degradation.

After a long delay in this project and consultation with technical support at both switch vendors, we revisited our testing after upgrading to the latest version of Mellanox OFED on the hosts. With the latest versions of firmware, drivers and utilities now available on the hosts, the RoCE performance on both switches suddenly improved tremendously. It is interesting to note that performance using PFC is excellent, while global pause flow control still exhibits poor performance. Our best guess is that some optimizations (or bug fixes) were made to the host PFC implementation which resulted in greatly improved RoCE performance. Note that precise hardware and software version information for our experimental setups is given in Table 3-1. While the performance results will be presented in the following sections, we will close this section by providing a detailed discussion of our RoCE configuration.

With the Connect-X5 NIC and Mellanox OFED, RoCE comes up when the device's kernel modules are loaded and is available without any additional configuration. The familiar IB utilities are available and working – for example `ibv_devinfo` shows a host channel adaptor (HCA) with Inifiniband transport but Ethernet link layer.

```
[carnac_host]# ibv_devinfo
hca_id: mlx5_0
        transport:                      InfiniBand (0)
        fw_ver:                         16.25.1020
        node_guid:                      ec0d:9a03:0048:7266
        sys_image_guid:                 ec0d:9a03:0048:7266
        vendor_id:                      0x02c9
        vendor_part_id:                 4119
        hw_ver:                         0x0
        board_id:                       MT_0000000011
        phys_port_cnt:                  1
        Device ports:
                port:   1
                        state:                  PORT_ACTIVE (4)
                        max_mtu:                4096 (5)
                        active_mtu:             4096 (5)
                        sm_lid:                 0
                        port_lid:               0
                        port_lmc:               0x00
                        link_layer:             Ethernet
```

And `show_gids` gives us a listing of identifiers for the HCA port. We chose index 3, corresponding to RoCE v2 and IPV4, for our testing.

```
[carnac_host]# show_gids
DEV     PORT    INDEX   GID                                     IPv4            VER     DEV
---     ----    -----   ---                                     ------------    ---     ---
```

```
mlx5_0  1       0       fe80:0000:0000:0000:ee0d:9aff:fe48:7266                 v1      eth1
mlx5_0  1       1       fe80:0000:0000:0000:ee0d:9aff:fe48:7266                 v2      eth1
mlx5_0  1       2       0000:0000:0000:0000:0000:ffff:0a64:0569 10.100.5.105    v1      eth1
mlx5_0  1       3       0000:0000:0000:0000:0000:ffff:0a64:0569 10.100.5.105    v2      eth1
n_gids_found=4
```

The only configuration that actually needs to occur on the host is enabling flow control. On Carnac, storage and management traffic goes over a dedicated network so the 100Gb "experiment" network only carries traffic related to our benchmarking. Since there is only one type of traffic there is no need to differentiate between priorities – we can use PFC and enable all priorities for flow control. We used PFC for all RoCE results that we report since we tested global pause flow control on the Arista switch and found poor performance (see Section 5).

With Mellanox OFED the `mlnx_qos` utility is used to enable PFC on the host interface.

```
[carnac_host]# mlnx_qos -i eth1 --pfc 1,1,1,1,1,1,1,1
DCBX mode: OS controlled
Priority trust state: dscp
dscp2prio mapping:
        prio:0 dscp:07,06,05,04,03,02,01,00,
        prio:1 dscp:15,14,13,12,11,10,09,08,
        prio:2 dscp:23,22,21,20,19,18,17,16,
        prio:3 dscp:31,30,29,28,27,26,25,24,
        prio:4 dscp:39,38,37,36,35,34,33,32,
        prio:5 dscp:47,46,45,44,43,42,41,40,
        prio:6 dscp:55,54,53,52,51,50,49,48,
        prio:7 dscp:63,62,61,60,59,58,57,56,
Receive buffer size (bytes): 262016,262016,0,0,0,0,0,0,
Cable len: 7
PFC configuration:
        priority    0   1   2   3   4   5   6   7
        enabled     1   1   1   1   1   1   1   1
        buffer      1   1   1   1   1   1   1   1
tc: 0 ratelimit: unlimited, tsa: vendor
priority:  1
tc: 1 ratelimit: unlimited, tsa: vendor
priority:  0
tc: 2 ratelimit: unlimited, tsa: vendor
priority:  2
tc: 3 ratelimit: unlimited, tsa: vendor
priority:  3
tc: 4 ratelimit: unlimited, tsa: vendor
priority:  4
tc: 5 ratelimit: unlimited, tsa: vendor
priority:  5
tc: 6 ratelimit: unlimited, tsa: vendor
priority:  6
tc: 7 ratelimit: unlimited, tsa: vendor
priority:  7
```

Likewise, PFC must be enabled on the switch. For the Arista EOS operating system an interface is configured for flow control on all priorities as follows.

```
ccore1(config)#priority-flow-control mode on
ccore1(config)#priority-flow-control priority 0 no-drop
ccore1(config)#priority-flow-control priority 1 no-drop
ccore1(config)#priority-flow-control priority 2 no-drop
ccore1(config)#priority-flow-control priority 3 no-drop
ccore1(config)#priority-flow-control priority 4 no-drop
```

```
ccore1(config)#priority-flow-control priority 5 no-drop
ccore1(config)#priority-flow-control priority 6 no-drop
ccore1(config)#priority-flow-control priority 7 no-drop
```

Similarly, for Mellanox Onyx PFC is enabled via the following commands.

```
100g-mlnx(config) # dcb priority-flow-control enable force
100g-mlnx(config) # dcb priority-flow-control priority 0 enable
100g-mlnx(config) # dcb priority-flow-control priority 1 enable
100g-mlnx(config) # dcb priority-flow-control priority 2 enable
100g-mlnx(config) # dcb priority-flow-control priority 3 enable
100g-mlnx(config) # dcb priority-flow-control priority 4 enable
100g-mlnx(config) # dcb priority-flow-control priority 5 enable
100g-mlnx(config) # dcb priority-flow-control priority 6 enable
100g-mlnx(config) # dcb priority-flow-control priority 7 enable
100g-mlnx(config) # dcb priority-flow-control priority 8 enable
100g-mlnx(config) # dcb priority-flow-control mode on force
```

Surprisingly, once the software stacks and firmware were brought up to the versions listed in Table 3-1 simply enabling PFC on the hosts and switches resulted in excellent RoCE performance for our small-scale testing.

5. RoCE Tuning

As previously discussed, a simple incast benchmark was useful in evaluating RoCE performance and recognizing poorly performing systems at small scale. A listing of the Python script used to perform this benchmark is provided in Appendix A.1. This benchmark is conceptually quite simple. The script shells into a collection of hostnames and starts the client/server benchmark executables as requested. However, since starting the remote shell can have both significant duration and variability it is important to run the benchmark for a long duration (5 seconds was used in this work). The sleep statements in the script have also proven critical to allow the subprocesses to start up and finish before moving on to the next stage of communication. An example invocation of this benchmark follows in which up to ten streams are sent from three round-robin hosts using an 8KB message size.

```
python traffic.py --test ib_write_bw --destination myhost1 --sources myhost[2,3,4] --options "-D 5 -s 8192"
```

The aggregate throughputs for each pair of source node and sending process numbers (each node can have multiple sending processes) can be plotted as seen in Figure 5-1, where results are shown for the Arista switch performing RoCE incast with PFC enabled. Good performance is indicated by high and consistent aggregate throughput, as seen for all message sizes greater than 1 byte. There is an increase in throughput variability with the large 1MB messages, but the throughput does not fall off significantly.

Contrasting with the good performance observed on the Arista switch using PFC, the results from the Arista switch using global pause flow control are shown in Figure 5-2. A significant difference between the PFC and global pause results is seen for the 1MB message size where incast from multiple source nodes using global pause results in a precipitous drop in performance as the number of sending processes increases. The global pause implementation is clearly not as successful as PFC in handling the congestion caused by the incast traffic. Similarly poor performance of global pause was observed when running the same tests over the Mellanox SN2700 switch. This benchmark also showed similar poor results for the earlier software/firmware versions that we tested on this hardware, but in this case the performance degradation was even more severe and observed at message sizes as small as 100KB. When we upgraded software/firmware we were easily able to observe the performance improvements using the incast benchmark with only 5 nodes as opposed to the hundreds of nodes needed to hit performance problems with the less demanding application benchmark that we were originally using for testing (High Performance Linpack).

The incast benchmark was also run at maximum transmission unit (MTU) sizes of 1024/1500 and 4096/4500 (IB/Ethernet – Ethernet must have a larger MTU than the IB protocol to allow

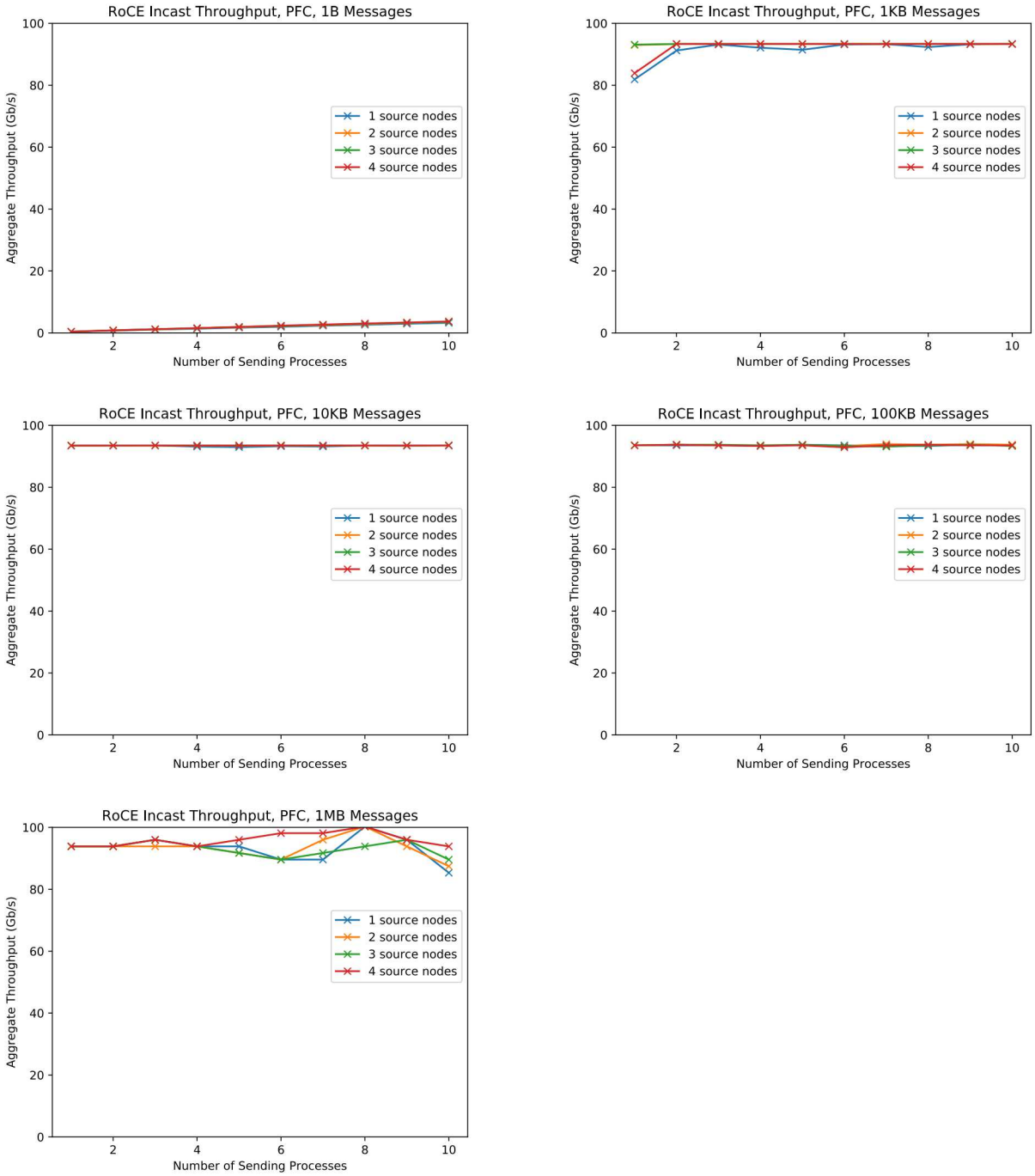

Figure 5-1. Aggregate throughputs showing good performance for incast testing of RoCE protocol on Arista DCS-7512N 100Gb/s Ethernet switch using priority flow control.

encapsulation). We determined that an MTU of 4096/4500 gave higher bandwidths without any observable trade-offs. All results that we report were obtained with an IB/Ethernet MTU of 4096/4500.

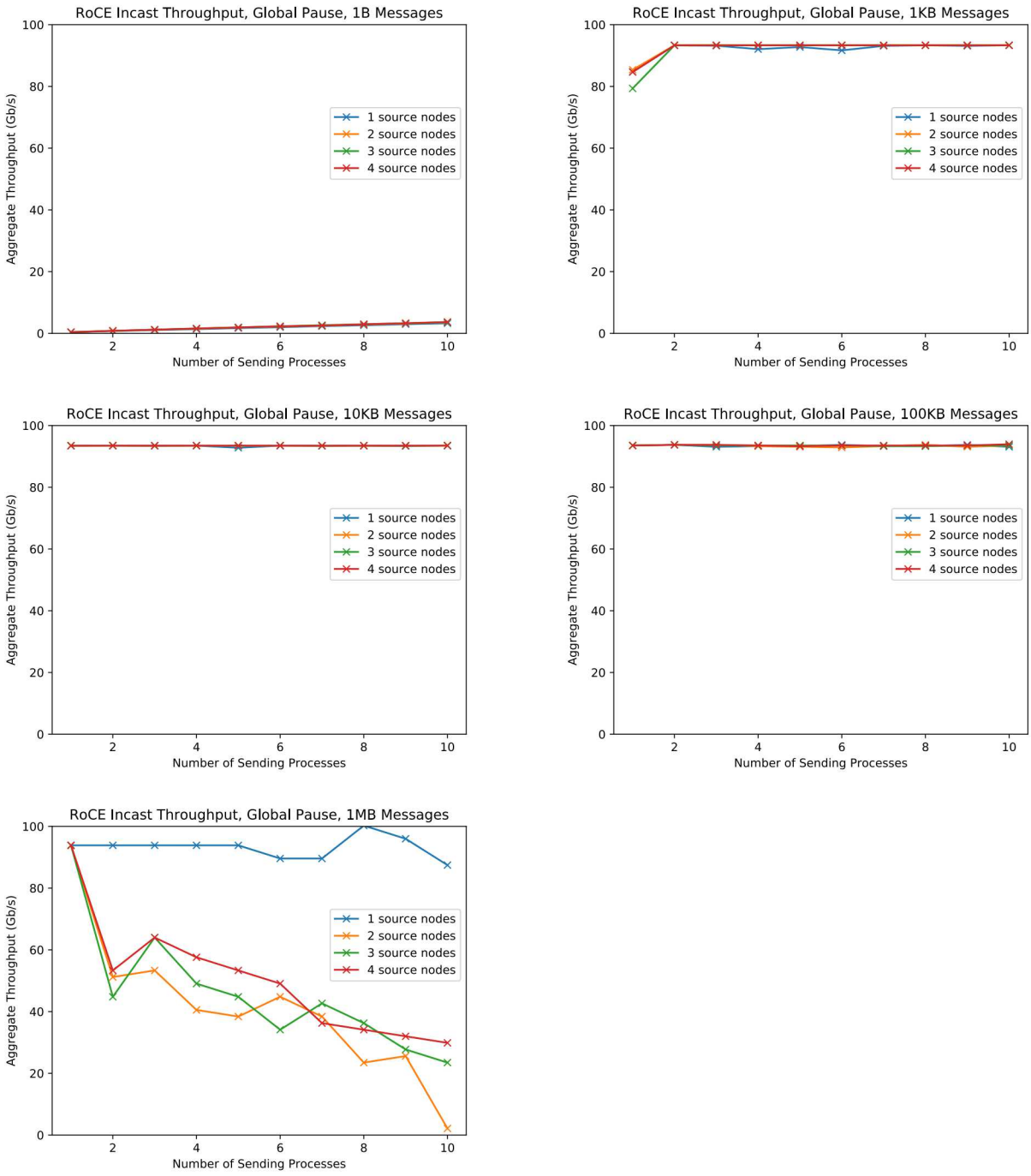

Figure 5-2. Aggregate throughputs showing poor performance of large messages for incast testing of RoCE protocol on Arista DCS-7512N 100Gb/s Ethernet switch using global pause flow control.

6. RoCE Performance Results

Once our network testbeds were configured, tuned and performing well as indicated by our incast benchmarks, we performed a limited set of small scale benchmarks to determine the value of testing RoCE more thoroughly at larger scales. In this section we report the results of MPI point-to-point bandwidth and latency tests, a comparison of TCP and RoCE performance for our incast benchmark, and small scale High Performance Linpack results. [1, 2] Since the focus of this work is RoCE, we did not spend time tuning the TCP stack. It is likely that significant gains in TCP performance could be obtained employing advanced Linux capabilities like traffic pacing, but here we compare "out-of-the-box" TCP protocols with RoCE.

Figure 6-1 presents large message bandwidths for MPI point-to-point messaging. We ran TCP tests with both a lossy (TCP) and lossless (TCP-PFC) fabric. Using RoCE protocols both switches attain near the nominal bandwidth of 100Gb/s while the best TCP results struggle to even approach 40Gb/s. Figure 6-2 reports small message MPI latencies. Again, we find very sizable performance improvements with RoCE protocols. While TCP latencies range from 12-14us, the Arista switch achieves latency under 3us with RoCE and the Mellanox switch attains a very impressive 1.3us, competitive with state-of-the-art HPC fabrics. Clearly, for point-to-point MPI messaging in the absence of interfering traffic, RoCE provides enormous performance benefits over TCP and can make Ethernet competitive with traditional high performance interconnects.

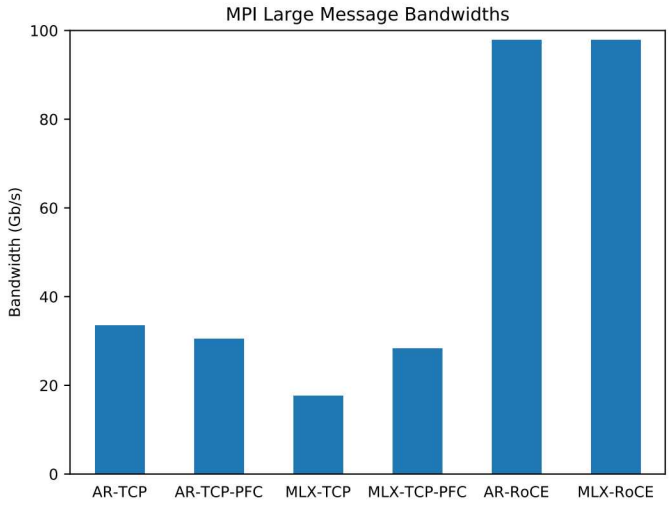

Figure 6-1. MPI point-to-point bandwidths for selected interconnect and protocol combinations using a message size of 4MB.

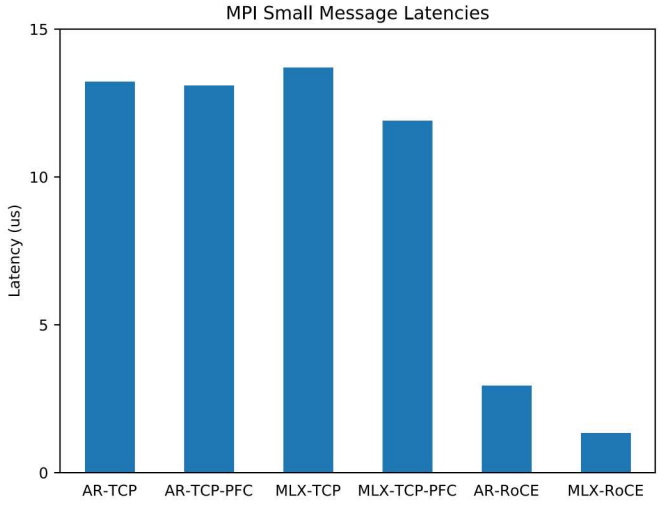

Figure 6-2. MPI point-to-point latencies for selected interconnect and protocol combinations using a zero-payload message.

Our incast benchmark script can generate roughly equivalent RoCE and TCP results using `ib_write_bw` and `iperf3`, respectively, as the underlying test executable (controlled by the `--test` parameter). This benchmark provides another way to contrast the performance of RoCE (Figure 6-3) and TCP (Figure 6-4) protocols. These figures were generated using the Mellanox SN2700 switch, and for RoCE the results are very much in line with those from the Arista switch – good performance with some variability at the largest message size. We see that TCP single stream bandwidths only reach about 20% of the 100Gb/s nominal bandwidth. Due to windowing protocols and the software overheads of TCP, low single-stream bandwidth is not surprising. [7] As the number of sending processes increases the aggregate throughput steadily increases to approximately 75% or more of nominal bandwidth.

Taken as a whole, the MPI point-to-point and incast results indicate that, for applications which require low latency or single streams of high bandwidth, RoCE should be a big win. However, as the number of communicating processes increases, as is likely with the increasing core counts of current CPU's, the bandwidth effects should have less impact. We conclude our performance results with Figure 6-5 which presents the FLOP/s performance of 32-node HPL computations. The HPL input can be found in Appendix A.2. We include results from the Serrano cluster (Omni-Path interconnect, see Section 3) for comparison with a typical HPC system and report percent of peak FLOP/s to normalize results. 32 MPI tasks were utilized on all systems and computation of peak FLOP/s was likewise based on 32 cores. As we were interested in relative network performance, we did not go to great lengths to optimize the on-node HPL performance and there is likely room to improve peak FLOP/s performance across the board. Though RoCE and Omni-Path perform slightly better than the TCP runs, the benefit is minimal. The small difference in performance is likely due to the small system scale, the large number of MPI tasks per node negating the RoCE bandwidth advantages, and the relatively tame resource demands of the HPL benchmark. RoCE is very competitive with Omni-Path performance for this test.

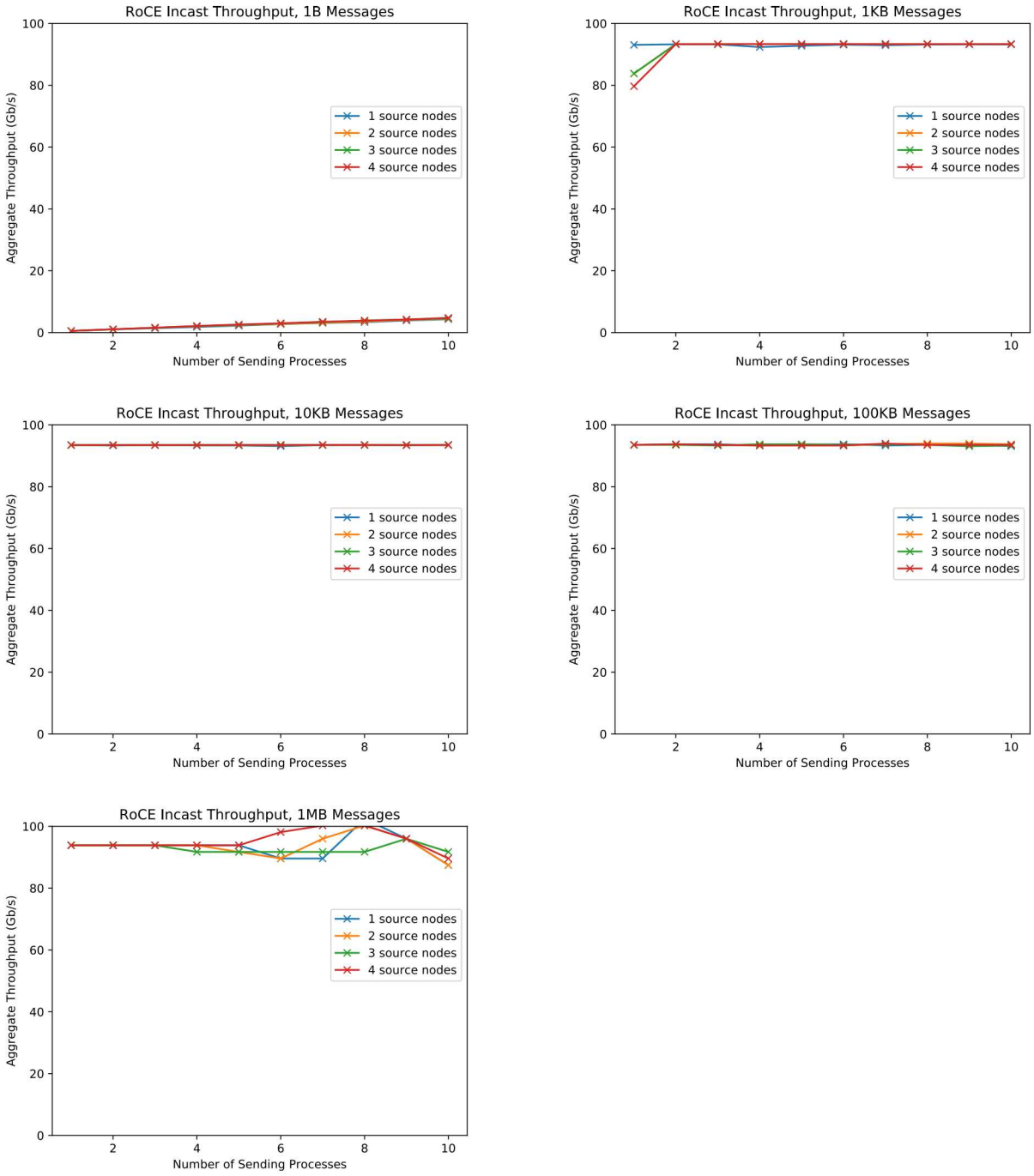

Figure 6-3. Aggregate throughputs for incast testing of RoCE protocol on Mellanox SN2700 100Gb/s Ethernet switch.

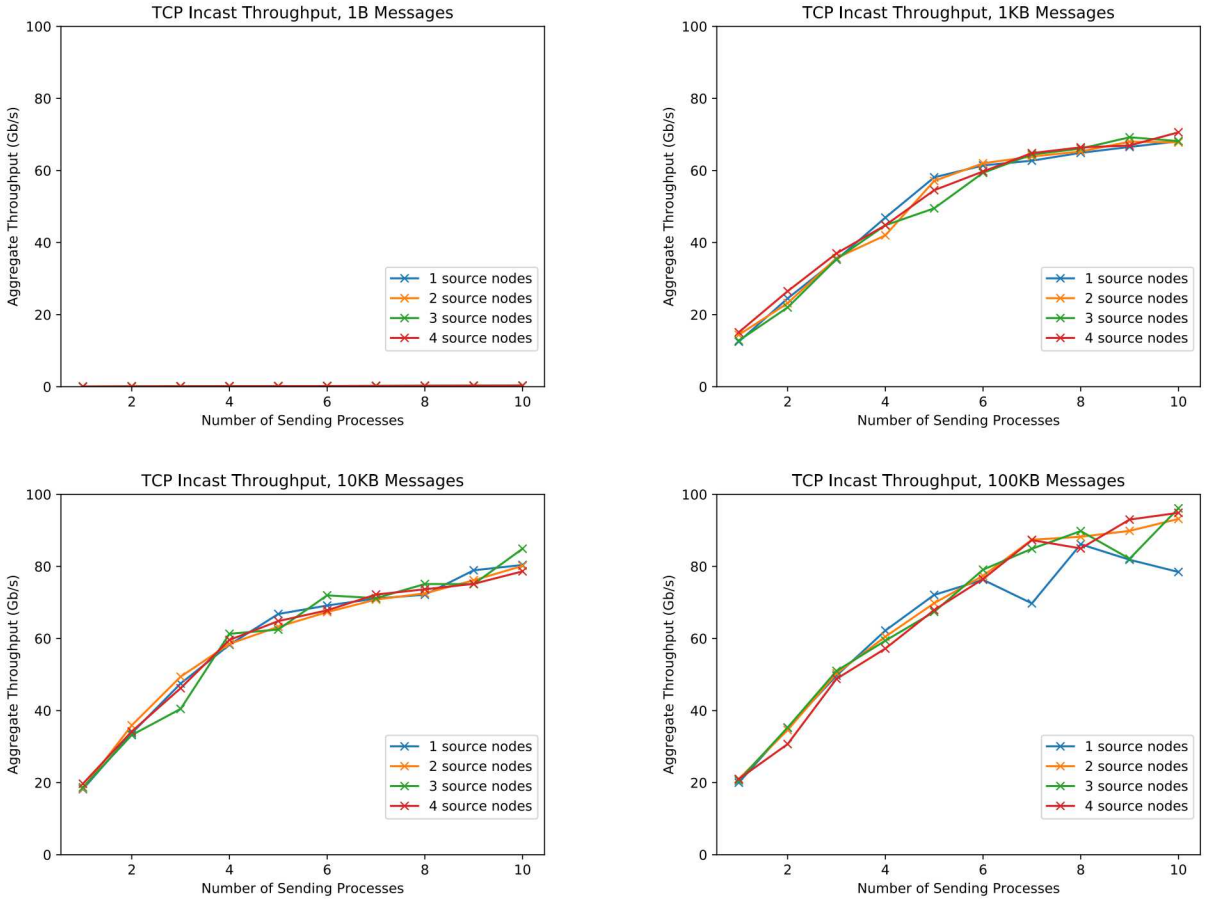

Figure 6-4. Aggregate throughputs for incast testing of TCP protocol on Mellanox SN2700 100Gb/s Ethernet switch.

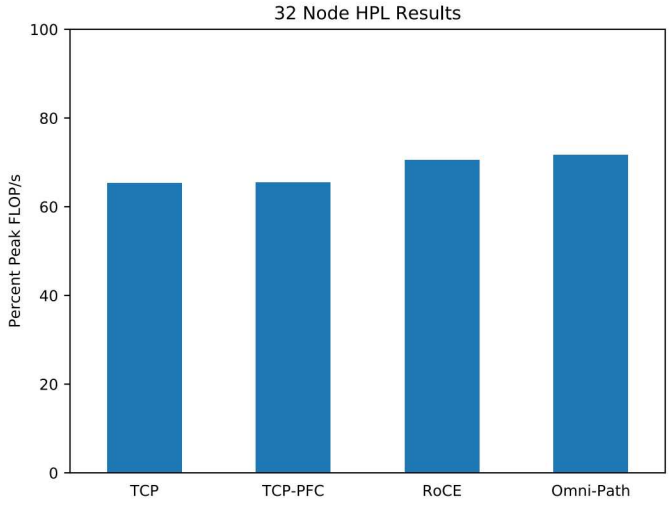

Figure 6-5. HPL peak efficiencies on Carnac using Mellanox switch (TCP, TCP-PFC and RoCE) and Serrano (Omni-Path).

7. Conclusions

Our results indicate that, at least at small scales, RoCE can provide improvements over TCP and is competitive with the performance of traditional HPC fabrics. Single stream MPI point-to-point benchmarks and a custom incast benchmark indicate that RoCE enables bandwidth and latency performance far exceeding that of "out-of-the-box" TCP protocols. However, our chosen proxy for HPC applications, HPL, does not show significant increased performance for RoCE, as it likely does not stress the available system resources sufficiently. Based on this initial evaluation, we feel that further study of RoCE for use on HPC systems, including the examination of a wider range of benchmarks and larger scales, is warranted.

While the configuration that eventually provided good performance was very basic, it was challenging to determine the proper configuration due to the complexity of Ethernet traffic management and a lack of comprehensive documentation. Coupling configuration difficulty with the immaturity of tools and software/firmware stacks severely complicated completion of this seemingly modest course of work. While consumer Ethernet products are mostly plug-and-play, large organizations that require sophisticated Ethernet networks devote significant resources to designing, configuring, testing, monitoring and maintaining their networks. While the performance of current RoCE implementations may make Ethernet networks viable for HPC systems, the likely increased costs of running an HPC Ethernet fabric must be factored into any procurement decisions.

References

[1] MVAPICH Home Page, 2019.

[2] Netlib HPL Webpage, 2019.

[3] Infiniband Trade Association et al. Supplement to infiniband architecture specification volume 1, release 1.2. 1: Annex a16: Rdma over converged ethernet (roce), 2010.

[4] Motti Beck and Michael Kagan. Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure. In *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*, pages 9–15. International Teletraffic Congress, 2011.

[5] J Floren, J Friesen, C Ulmer, and S T Jones. *A Reference Architecture for Emulytics Clusters*. In *Sandia Report*, volume SAND2009-5574, 2017.

[6] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 202–215. ACM, 2016.

[7] Justin Hurwitz and Wu-chun Feng. Initial end-to-end performance evaluation of 10-gigabit ethernet. In *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, pages 116–121. IEEE, 2003.

[8] Michael Oberg, Henry M Tufo, Theron Voran, and Matthew Woitaszek. Evaluation of rdma over ethernet technology for building cost effective linux clusters. In *7th LCI International Conference on Linux Clusters: The HPC Revolution*, 2006.

[9] Jerome Vienne, Jitong Chen, Md Wasi-Ur-Rahman, Nusrat S Islam, Hari Subramoni, and Dhabaleswar K Panda. Performance analysis and evaluation of infiniband fdr and 40gige roce on hpc and cloud computing systems. In *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*, pages 48–55. IEEE, 2012.

Appendix A.

A.1. Incast Python Code Listing

```
from __future__ import print_function

import sys
import argparse
import subprocess
import time
import re
from sets import Set

def get_ids(hostnames):
  hostnames = "".join(hostnames.split())
  brckt = hostnames.count("[")
  hosts = []
  if not brckt:
    spl = hostnames.split(",")
    for h in spl:
      hosts.append(h)
  else:
    spl = hostnames.split("[")
    basename = spl[0]
    ids = spl[1].rstrip("]")
    if ids.count(",") > 0:
      ids = ids.split(",")
    # deal with ranges
    for i in ids:
      if i.count("-") == 0:
        hosts.append(basename + i)
      else:
        rng = i.split("-")
        for n in range(int(rng[0]),int(rng[1])+1):
          hosts.append(basename + str(n))
  return hosts

def main():

  parser = argparse.ArgumentParser(description="Traffic Benchmark")
  parser.add_argument("--destinations", default="", help="destination hostnames in slurm format")
  parser.add_argument("--sources", default="", help="source hostnames in slurm format")
  parser.add_argument("--outfile", default="traffic.out", help="output filename")
  parser.add_argument("--max-streams", default="10", help="max number of streams (default 10)")
  parser.add_argument("--options", default="", help="additional arguments to test program")
  parser.add_argument("--client-options", default="", help="additional arguments to test client")
  parser.add_argument("--server-options", default="", help="additional arguments to test server")
  parser.add_argument("--port", default=11000, type=int, help="first in sequential block of ports used")
  parser.add_argument("--test", default="ib_write_bw", help="should be ib_write_bw, ib_send_bw or iperf3")
  args = parser.parse_args()

  buffsize=0
  outf = open(args.outfile,'w',buffering=buffsize)

  clients = get_ids(args.sources) # clients do the sending
  servers = get_ids(args.destinations)
```

```
  nclt = len(clients)
  nsrv = len(servers)
  maxn = max(nclt,nsrv)
  optstr = args.options
  portbase = args.port
  ptest = args.test
  outf.write("nsrc=%d ndest=%d\n" % (nsrv, nclt))
  outf.write("test=%s\n" % ptest)
  outf.write(optstr)

  for nstreams in range(1,int(args.max_streams)+1):
    print("starting with %d streams" % nstreams)
    outf.write("*** nstreams: %d ***\n" % nstreams)
    srvp = []
    cltp = []

    # start servers
    for i in range(1,nstreams+1):
      port = portbase + int(i)
      server = servers[(i-1) % nsrv]
      my_args = ['ssh', server, ptest, '-p %d' % port]
      my_args.append(optstr)
      my_args.append(args.server_options)
      if ptest == "iperf3":
        my_args.append("-s --one-off")
      print("server %s ready to receive on port %d" % (server,port) )
      outf.write(' '.join(my_args))
      outf.write('\n')
      srvp.append(subprocess.Popen(my_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE))

    # give servers a chance to start up
    time.sleep(5.0)

    #start clients
    for i in range(1,(nstreams)+1):
      port = portbase + int(i)
      client = clients[(i-1) % nclt]
      server = servers[(i-1) % nsrv]
      my_args = ['ssh', client, ptest, '-p %d' % port]
      my_args.append(optstr)
      my_args.append(args.client_options)
      if ptest == "iperf3":
        my_args.append("-c %s" % server)
      else:
        my_args.append(server)
      print("client %s sending on port %d" % (client,port) )
      outf.write(' '.join(my_args))
      outf.write('\n')
      cltp.append(subprocess.Popen(my_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE))

    # wait for completion
    for p in srvp:
      p.wait()
    for p in cltp:
      p.wait()
      out, err = p.communicate()
      outf.write(out)
      outf.write(err)
    time.sleep(5.0)

if __name__ == "__main__": main()
```

A.2. High Performance Linpack Input

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1 #4            # of problems sizes (N)
262144 # (256 * 32 * 32)
1            # of NBs
256 #1 2 3 4      NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1 #3            # of process grids (P x Q)
32 #16 #2 1 4        Ps
32 #20 #2 4 1        Qs
16.0         threshold
1 #3            # of panel fact
0 #0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
1 #2            # of recursive stopping criterium
2 #2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1 #3            # of recursive panel fact.
0 #0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Distribution

Hardcopy—External

Number of Copies	Name(s)	Company Name and Company Mailing Address

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.