Zettar zx Evaluation for ESnet DTNs

Report on findings, November 2020

Contributors: Ezra Kissel, ESnet[1], Chin Fang, Zettar Inc.[2] (Zettar)

[1] “Energy Sciences Network”. https://es.net/. Accessed 20, Nov, 2020.

[2] “Zettar - Moving Data at Scale and Speed”. https://zettar.com/. Accessed 20, Nov, 2020.

1 Overview

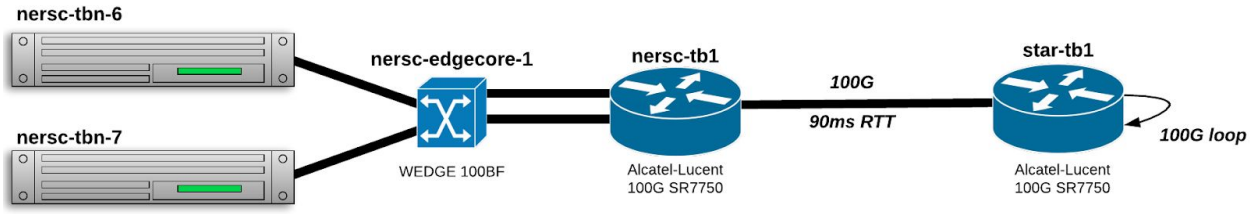
ESnet is prototyping a Data Transfer Node as-a-Service (DTNaaS) capability that aims to provide optimized, on-demand data movement tools/endpoints to users of the network. Zettar offers a high-performance data movement solution, `zx`[3], that integrates with a number of storage technologies and provides mechanisms for API automation. An evaluation of the solution within the ESnet testbed environment was performed over the duration of approximately 2 months. The performance of disk I/O and network interactions were explored in a containerized software environment.

2 Testbed Resources

The testing was performed within the ESnet 100G testbed[4] where the DTNaaS prototype is being developed. Two GIGABYTE R281-NO0-00[5] servers (DTN servers `nersc-tbn-6/7`) were used with the following specifications.

2x 12-core Intel(R) Xeon(R) Gold 6146 CPU @ 3.20GHz[6]
384GB RAM
8x NVMe 3.84TB SSDs in Redundant Arrays of Independent Disks (RAID)-0 (Intel VROC[7]) [HGST Ultrastar SN200 Series NVMe SSD[8]]
Mellanox ConnectX-5 dual-port 100G NICs[9]

[3] "Products". https://www.zettar.com/our-products/. Accessed Nov. 26, 2020.
[4] "100G SDN Testbed". https://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed/. Accessed 20, Nov, 2020.
[5] "R281-NO0". https://www.gigabyte.com/us/Rack-Server/R281-NO0-rev-400. Accessed 20, Nov, 2020.
[6] "Intel® Xeon® Gold 6146 Processor". https://ark.intel.com/content/www/us/en/ark/products/124942/intel-xeon-gold-6146-processor-24-75m-cache-3-20-ghz.html. Accessed 20, Nov, 2020.
[7] "Intel® Virtual RAID on CPU (Intel® VROC)". https://www.intel.com/content/www/us/en/support/articles/000024498/memory-and-storage/ssd-software.html. Accessed 20, Nov, 2020.
[8] "Western Digital Ultrastar SN200 NVMe SSD Data Sheet". https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/data-center-drives/ultrastar-dc-ha200-series/data-sheet-ultrastar-dc-sn200.pdf. Accessed 20, Nov, 2020.
[9] "ConnectX®-5 EN Single/Dual-Port Adapter Supporting 100Gb/s Ethernet". https://www.mellanox.com/products/ethernet-adapters/connectx-5-en. Accessed 26, Nov, 2020.



The NERSC[10] DTNs are connected through a local switch but may also make use of a higher latency WAN lookback path through Starlight[11] using tagged interfaces pre-configured on each host. The diagram above shows the logical connectivity used over the course of the evaluation period.

3 Science DMZ

In anticipation of the high data rates, the testbed setup has a similar environment as one would find in a Science DMZ, which was formulated by Eli Dart, ESnet in 2010[12]. The ubiquitous firewall, although the workhorse device of network security, has a poor track record in high-performance context[13]. Thus, the network paths used in the testbed do not include a firewall device (beyond host-based filtering) and the testbed paths were engineered to be as free of loss as possible.

4 System and Software Environment

Running Zettar `zx` in a containerized environment was the primary usage model being investigated. CentOS 8[14] was chosen as the OS environment for the host. Podman[15] was considered but not ultimately used as there had been existing investment in configuring and tuning Docker containers for DTNaaS.

```
CentOS Linux release 8.2.2004 (Core)
Kernel 4.18.0-147.8.1.el8_1.x86_64
MLNX_OFED_LINUX-5.0-2.1.8.0-rhel8.1-x86_64
Docker version 19.03.13, build 4484c46d9d
```

[10] "National Energy Research Scientific Computing Center". https://www.nersc.gov. Accessed 20, Nov. 2020.
[11] "STARLIGHT. The Optical STAR TAP℠". http://www.startap.net/starlight/. Accessed 20, Nov. 2020.
[12] "ESnet's Science DMZ Breaking Down Data Barriers, Speeding up Science". https://www.es.net/about/esnet-history/esnets-science-dmz-breaking-down-data-barriers-speeding-up-science/. Accessed 29 Nov. 2020.
[13] "Building A Science DMZ". https://www.es.net/assets/pubs_presos/20130113-dart-Science-DMZ2.pdf (slide 14). Accessed 29 Nov. 2020.
[14] "The CentOS Project". https://www.centos.org. Accessed 24, Nov. 2020
[15] "The Pod Manager Tool". https://podman.io. Accessed 20 Nov. 2020.

4.1 Docker containers

The Zettar `zx` containers were built from the centos/systemd[16] container available on DockerHub[17]. CentOS 7 was selected as the OS environment for the containers. The containers were launched with host networking and in privileged mode (i.e. using `--net host` and `--privileged` flags). The VROC RAID filesystem was mounted inside the container using a bind volume with an available capacity of 23TB. Each container had full use of all CPU cores and all available memory on the host.

CentOS Linux release 7.8.2003 (Core) Filesystem: XFS

4.2 Host and network tuning

Host tuning was performed using the DTNaaS tuning agent that sets appropriate `sysctl` parameters for high-throughput TCP networking (see **Appendix** for details).

The Mellanox network adapters were tuned using the `mlnx_tune`[18] utility and the `HIGH_THROUGHPUT` option[19]. The Linux interfaces were configured with 9000 byte MTUs and a `tx-queuelen` of 10000. Intel Input–output memory management unit (iommu)[20] and SR-IOV[21] were enabled with 8 VF[22]s for each NIC but not used directly for this testing.

The CPU frequency governors were set to the performance profile on each host.

The `nofile` ulimit[23] was increased from the default 1024 to 131072 in each container. No additional tuning was performed on the NVMe or VROC subsystems.

4.3 Hyper-threading

`nersc-tbn-6` has Intel hyper-threading enabled while `nersc-tbn-7` has hyper-threading disabled. This configuration resulted in `tbn-6` having 48 logical cores while `tbn-7` had 24

---

[16] "centos/systemd - Docker Hub." https://hub.docker.com/r/centos/systemd/. Accessed 20 Nov. 2020.
[17] "Docker Hub". https://hub.docker.com. Accessed 20 Nov. 2020.
[18] "Mellanox userland tools and scripts". https://github.com/Mellanox/mlnx-tools. Accessed 20 Nov. 2020.
[19] "mlnx-tools/ofed_scripts/utils/mlnx_tune". https://github.com/Mellanox/mlnx-tools/blob/master/ofed_scripts/utils/mlnx_tune. Accessed 24 Nov. 2020.
[20] "Input–output memory management unit". https://en.wikipedia.org/wiki/Input–output_memory_management_unit. Accessed 24 Nov. 2020.
[21] "Single-root input/output virtualization". https://en.wikipedia.org/wiki/Single-root_input/output_virtualization. Accessed 24 Nov. 2020
[22] "Virtual network interface". https://en.wikipedia.org/wiki/Virtual_network_interface. Accessed 24 Nov. 2020
[23] "bash(1) - Linux manual page". https://man7.org/linux/man-pages/man1/bash.1.html. Accessed 24 Nov. 2020.

logical cores. This configuration was intentionally left in place to observe the differences in `zx` behavior in each logical core count scenario.

5 Disk I/O benchmarking

At the beginning of any serious moving data at scale and speed project, the first step is always to carry out disk I/O benchmarking. This is simply due to the fact that it's the ultimate limiting factor to the attainable data movement throughput - data is always pulled first from and drained finally to storage.

Furthermore, it is imperative to employ a test data set consisting of data files/objects that reflect production data - the file size histogram of a dataset can impact the attainable data movement rates significantly. Lacking such a dataset, the next best thing is to conduct a "sweep". See the **Results** section below. The results from such a "sweep", in conjunction with a file size histogram and a proper scaling to account for file storage differences, can be very helpful to "back-of-the-envelope" estimates for new situations. See **7.2** and **8.2.3** under **Robustness testing**.

A properly conducted disk I/O benchmarking sets the correct expectations for everyone.

In carrying out disk I/O benchmarking for an I/O intensive software application, there are three principles to follow:

1. Simulating the storage interactions of the application as much as possible using the benchmarking tool. Said interactions include factors such as I/O patterns, concurrency, data chunk sizes, using or bypassing the Linux OS pagecache or not. Only after such an understanding of the application is attained, is it time to configure the disk I/O benchmarking tool.
2. Always carrying out the disk I/O benchmarking in the storage client nodes, never directly on the storage nodes themselves. The disk I/O seen by an application is the one measured in the application node, which is most likely a storage client.
3. For a cluster oriented application running on X number of nodes, the disk I/O benchmarking should be conducted concurrently on these X number of nodes, not one-by-one.

Two tools are used: `fio`[24] and `elbencho`[25]. The former has been used as the official QA tool of the Linux kernel's block subsystem since 2005[26], but has been extended over the years to cover many other use cases. The creation of `elbencho` was inspired by traditional storage benchmark tools like `fio`, `mdtest` and `ior`[27], but was written from scratch to replace them with

[24] "Flexible I/O Tester". https://github.com/axboe/fio. Accessed 20 Nov. 2020.
[25] "A distributed storage benchmark for file systems and block devices with support for GPUs". https://github.com/breuner/elbencho. Accessed 22 Nov. 2020.
[26] "Releases.axboe/fio.GitHub". https://github.com/axboe/fio/releases?after=fio-1.3. Accessed 20 Nov. 2020.
[27] "IOR and mdtest parallel I/O benchmarks". https://github.com/hpc/ior. Accessed 20 Nov. 2020.

a modern and easy to use unified tool for file systems and block devices. During this project, it has been proven to be valuable in cases that are highly challenging or even impossible to tackle using `fio`.

`elbencho` is simple to use and quick to run. It is new and thus may not cover some cases conveniently. But it's also being rapidly improved. So, for moving data at scale and speed, we anticipate that `elbencho` will become the "go-to" tool to use for disk/IO benchmarking.

6 Zettar/`zx` installation, configuration, and tuning

6.1 `zx` installation and configuration

Zettar provides extensive software installation and configuration instructions on its online documentation site[28]. The Zettar team worked closely with ESnet to install the necessary software packages (RPMs) within the containers and configure the `zx` service itself. A brief summary of steps is outlined below:

1. Received credentials to access the Zettar `yum` repository.
2. Used `yum` to install `zx-core` and `zx-pysdk` packages.
3. Received a Zettar `zx` evaluation license file. This file is installed in the `zx` configuration path. Without a valid license file the `zx` service will not start.
4. Ran the interactive `zx_config.sh` configuration wizard, a `ncurses` utility that prompts for common configuration parameters and writes a `zx` configuration file that can be used for the `zx` service.
5. Enabled and started the `zx` service using `systemctl`.

Once started, the `zx` built-in webUI may be accessed with a browser. Using this interface, new sites may be added. For this evaluation, the webUI was used to add the remote site (`tbn-6/7`) as appropriate using the IP addresses of the tagged WAN loop interfaces on each host.

The Zettar team uses Ansible[29] to manage `zx` deployment and configuration.

6.2 `zx` tuning

After initial installation, the Zettar team began a series of tuning steps to achieve consistent disk-to-disk transfer performance between the two evaluation hosts. This included both disk I/O benchmarking to understand the behavior of the NVMe drives in use, and benchmarking `zx` transfers over the network to understand the characteristics of the 100G WAN path.

---

[28] "Zettar documentation". https://docs.zettar.com (*access controlled*). Accessed 20 Nov. 2020.
[29] "Ansible is Simple IT Automation". https://www.ansible.com/. Accessed 20 Nov. 2020.

The configuration options that were eventually arrived at and used for the `zx` ***non-TLS encrypted*** **Transfer "sweep"** results are presented in the **Appendix**.

7 Testing methodology

ESnet developed a testing script that used the `zx` Python SDK (REST API client library) to start/stop `zx` transfers and monitor their status. The output of the status call is a JSON object that contains transfer details including bytes transferred and total task time from which the effective transfer rate can be calculated. Checksumming is enabled and unconditional for all zx transfers.

Note that `zx` is symmetric in nature. In other words, it is not of the traditional client/server architecture. Once two `zx` instances have been paired-up, any instance of `zx` can send/receive with the other one, even concurrently. `zx` can be fully controlled using its REST APIs without involvement of the built-in webUI. Zettar's `zx-File` product was used for all `zx` tests - note that it is only one of Zettar software products that `zx` integrates.

Globus Connect Server v4 (GCSv4) was used to represent a widely-used file transfer tool in comparison to `zx`. A separate container image with the GCSv4 software was built and run alongside the `zx` container on each host. Parallel TCP streams, concurrency, and pipelining options were all employed as appropriate for each data set transferred as described below. `globus-url-copy` was used as the client utility for invoking each 3rd-party transfers between source and destination servers.

7.1 Transfer "sweep"

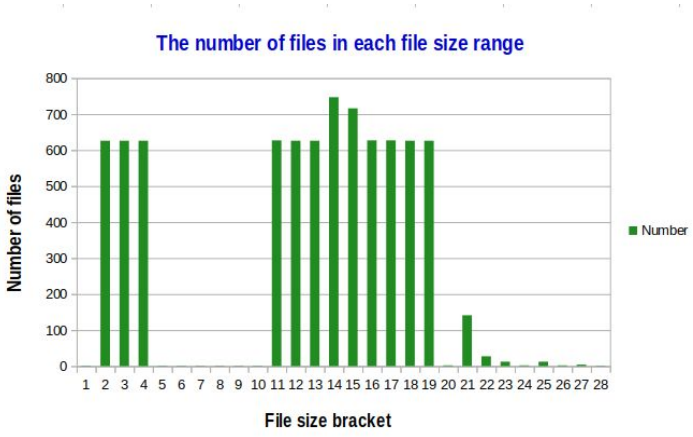
A number of data sets were generated to explore how the transfer service deals with file distributions: LOSF (*1KiB ≤ file size < 1MiB files*), medium sized files (*1MiB ≤ file size < 1GiB*), and large files (*1GiB ≤ file size ≤ 1TiB*). Then, automated "sweeps" were conducted.

7.2 Robustness testing

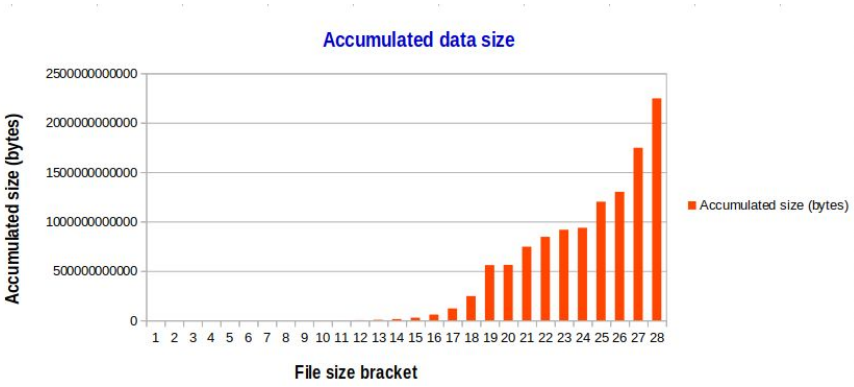
Furthermore, a sample production dataset from the ESnet DTN testing endpoints was retrieved. The dataset is 2.1TB in size. It consists of mix-sized files. The storage throughput measured for this dataset using `fio` is given below (*`fio` can benchmark such datasets, `elbencho` not yet*). Please see the `fio` job file in the **Appendix**.

	1st run	2nd run	3rd run	Mean	Median	Std dev
Throughput (Gbps)	82.4	80.0	84.0	82.67	82.4	2.01

The histogram of the dataset is graphically shown. The textual output is given in the **Appendix**



Then, the dataset is transferred repeatedly until 1PB is reached. Both non-TLS and TLS encrypted modes were used. Owing to the long duration and the way the test dataset is used, both the TLS and non-TLS transfer are only carried out once. To shorten the overall duration for each run, `zx` settings are tuned specifically for the test dataset, in non-TLS and TLS modes. The results were presented in the **Results** section. The `zx` settings used are given in the **Appendix**.



7.3 Monitoring

Each node (on host OS) was running Prometheus[30] node_exporter and hsflowd to provide some minimal level of instrumentation. A set of Grafana[31] dashboards were used to inspect and verify network traffic at a per-interface granularity. The `nersc-edgecore-1` switch was also exporting sflow[32] records to the sflow-RT to provide additional data points for per-port performance counters.

8 Results

Results were collected using the ~90ms WAN loop configured between NERSC and Starlight. Software versions used:

zx 4.1.4145 a17426504c8ef16cc428 2020-11-05 12:40:14 (OpenSSL 1.1.1h 22 Sep 2020) globus_gridftp_server: 12.24 (1590108598-85) (globus connect server v4) elbencho Version: 1.6-0 (*the latest master branch is always compiled and used*)

[30] "Prometheus - Monitoring system & time series database". https://prometheus.io. Accessed 20 Nov. 2020.
[31] "Grafana: The open observability platform". https://grafana.com. Accessed 20 Nov. 2020.
[32] "Making the Network Visible". https://sflow.org. Accessed 20 Nov. 2020.

8.1 Transfer “sweep”

Only hyperscale datasets were used for the sweep, where the term “hyperscale data” is defined as overall size ≥ 1TB, or number of files ≥ 1 million, or both. As an example, the overall size of the dataset 1048576x512MiB is 1TiB = 1.099TB. There are 1048576 files and 1048576 ≥ 1,000,000 (1 million). A flat directory layout is used to simulate the common non-ideal cases as seen in practice.

The GCSv4 parameters for the “sweep” were adapted for three regions across the data sets transferred. For all 1 million file sets, concurrency was set to 32 and pipelining enabled. For 1MiB to 1GiB sets, concurrency was set to 24 and pipelining enabled. For 2GiB to 1TiB, concurrency was set to 12 and pipelining was disabled. For the data sets with larger files, it was observed that having pipelining enabled prevented max concurrency as the transfer progressed leading to slower tail-end transfer rates. At the other end of the spectrum, concurrency at 32 and pipelining provided the best performance for the lots of small file (LOSF) cases. We note that only a single configuration profile was used for the `zx` testing in comparison.

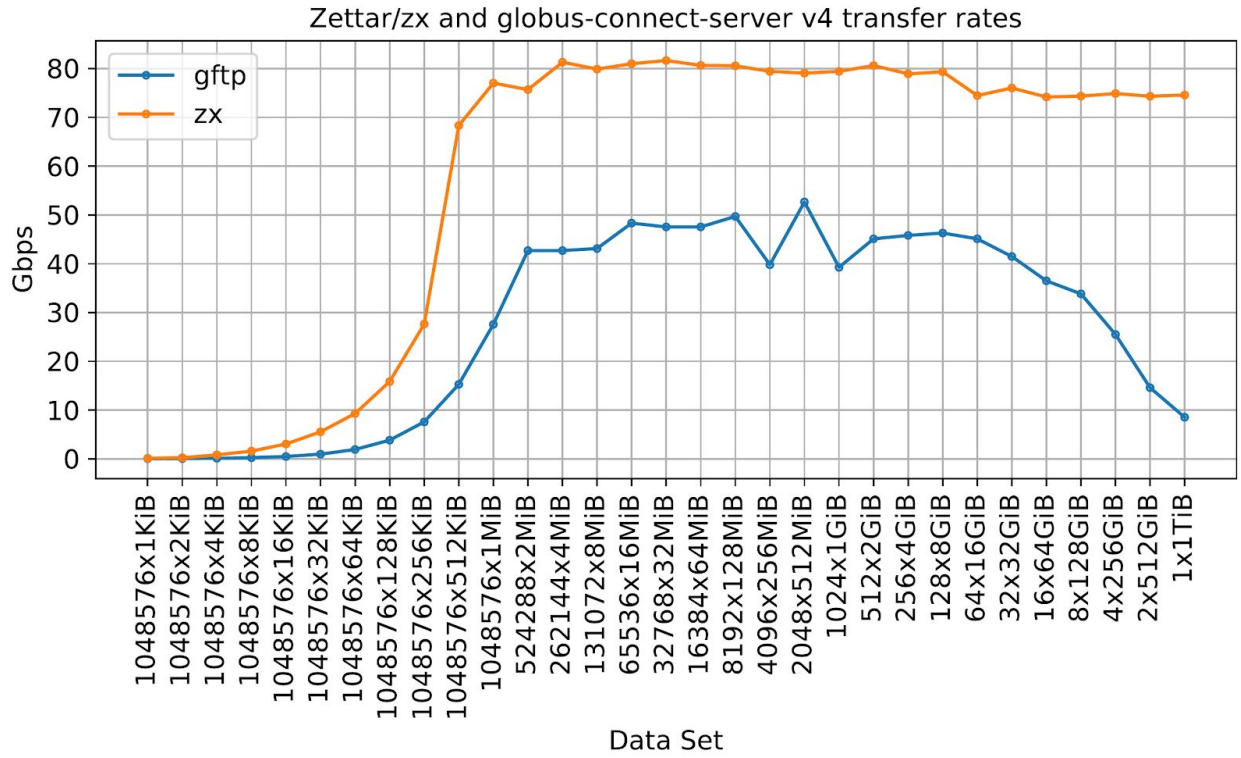


***Fig. 1**: `zx` and GridFTP disk-to-disk transfer rate from tbn-7 to tbn-6. Each data set is named according to the convention: number_of_files X the common size of all files. So, 2x512GiB means there are 2 files in this dataset and each file is of the common size 512GiB*

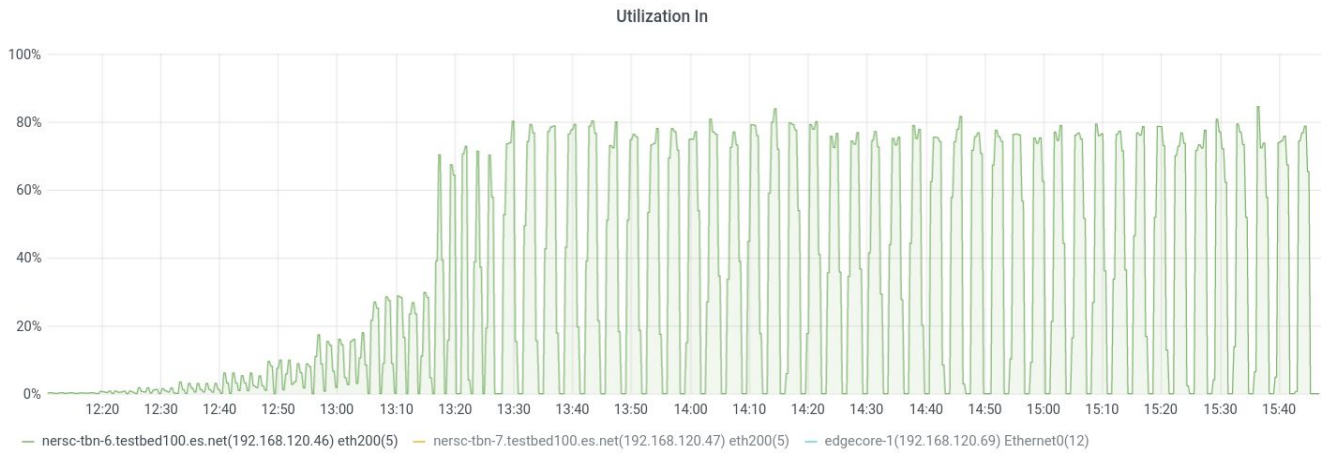


***Fig. 2**: sflow monitoring of tbn-6 ingress interface, tracking performance of zx service results*

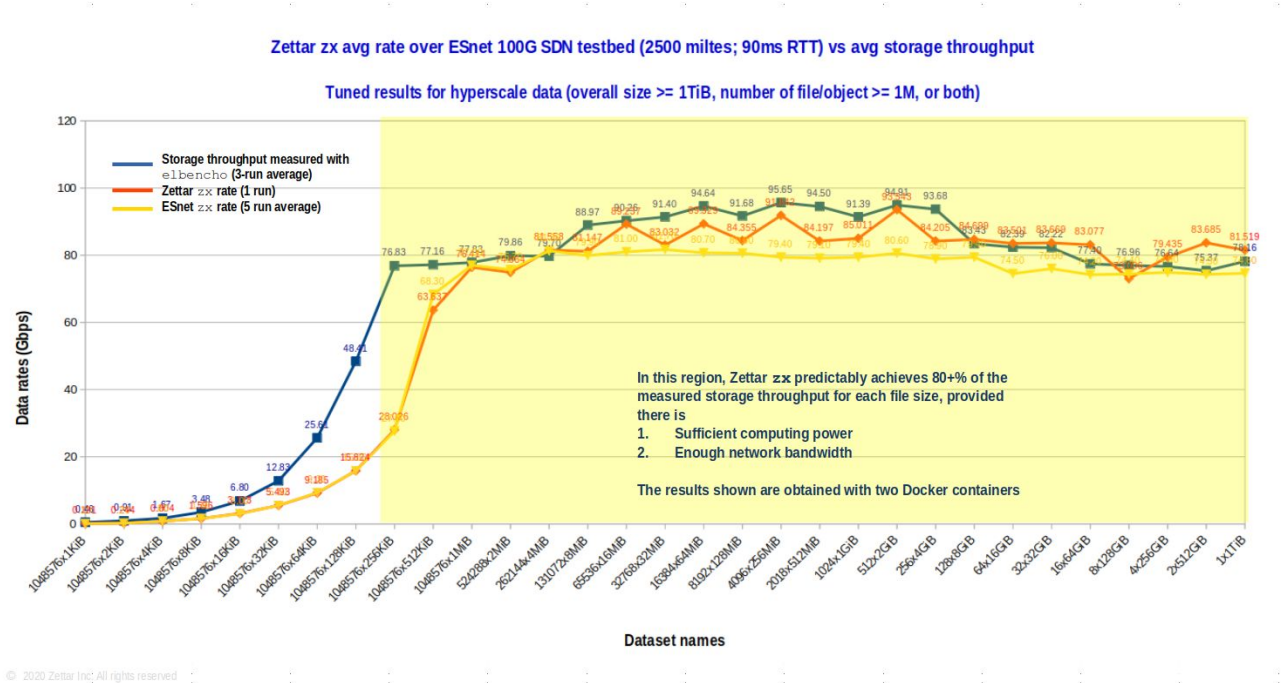


***Fig. 3**: A combined line chart with the measured storage throughput for each file size (**blue line**), together with both the Zettar `zx` transfer data rates attained with a single run carried out by Zettar (**orange line**), and the average of five runs carried out by ESnet (**yellow line**)*

8.2 Robustness testing

7.2.1 Motivations

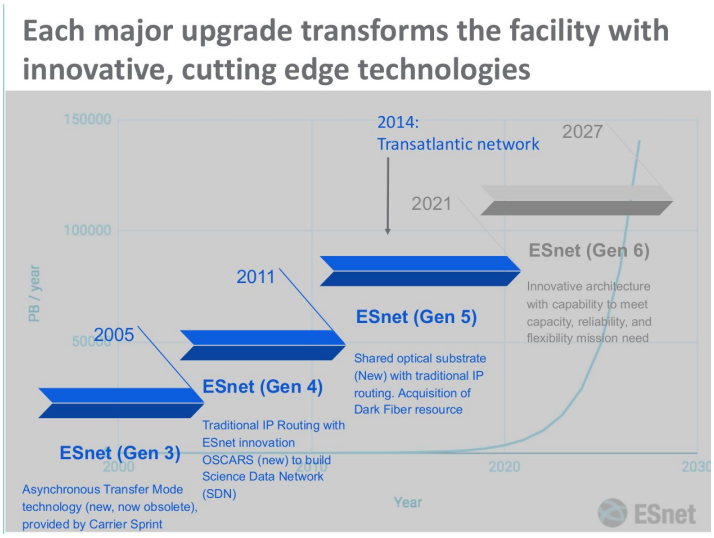


Fig. 4*: Total data traffic handled by ESnet and recent ESnet generations*

The massive volumes of data in PB/year handled by the ESnet[33] is shown on the left.

With such high volumes and the typical long durations associated with moving massive datasets across various links operated by ESnet, it's essential to investigate the robustness of the set up, even during the prototyping stage. The following testing methodology is adopted from the SLAC/Zettar high-speed data transfer research[34] that spanned from 2015 - 2019: always transfer 1PB data over WAN in both TLS and non-TLS modes.

8.2.2 TLS transfer of 1PB

The final mean transfer speed is 53.92 Gbps. This is *only* 65.22% of the measured mean storage throughput 82.67Gbps as reported in the **Testing methodology** section. The total transfer duration is 41 hr 11 min 39 sec. Three possible causes of the low rate achieved are discussed in the **Appendix**.

Fig. 5*: The sFlow-RT Network Interfaces Grafana dashboard display of the TLS transfer of 1PB over the 100G testbed.*

[33] "ESnet and ESnet6 project: Overview". Inder Monga. Director, Energy Sciences Network. Division Director, Scientific Networking, Lawrence Berkeley National Lab. DOE booth. Supercomputing 2018

[34] "Next Generation high performance, multi-dimensional scalable data transfer". https://indico.cern.ch/event/505613/contributions/2230905/. Accessed November 23, 2020

8.2.3 Non-TLS transfer of 1PB

The final mean transfer speed is 77.25 Gbps. This is 93.44% of the measured mean storage throughput 82.67 Gbps as reported in the **Testing methodology** section. The total transfer duration is 28 hr 45 min 15 sec. The results correlate with **FIG. 3** well.

***Fig. 6**: The sFlow-RT Network Interfaces Grafana dashboard display of the non-TLS transfer of 1PB over the 100G testbed.*

Together with the TLS transfer of 1PB, the two tests have shown that the main goal of the robustness testing is achieved.

9 Additional project accomplishments

Other than the main project goal, we have accomplished the following:

1. Bench-marked attainable storage throughput using `elbencho` from 1KiB to 1TiB and published the results.
2. Demonstrated that modern TCP, as long as it's used correctly and in low-loss environments, can be very efficient for bulk data transfer over long distances. The often-heard assertion that UDP (*or proprietary protocols*) is better than TCP for such use cases is questionable[35].
3. Confirmed that with a well-provisioned infrastructure, just with data mover software alone, it's feasible to use a single setting to cover a wide range of file sizes at 80% or more of the corresponding measured storage throughput. Using proprietary add-on hardware is unnecessary.
4. Showed that with properly provisioned and configured Docker containers, it's feasible to use them for moving data at scale and speed with close to "bare-metal" performance, for both TLS encrypted and unencrypted transfers.

[35] "UNIX Network Programming Networking APIs: Sockets and XTI, Volume 1, 2nd Ed, by W. Richard Stevens, p 563" specifically warned that "UDP can be used for simple request-reply scenarios, but some form of reliability must then be added to the application. UDP should not be used for bulk data transfer."

5. Exemplified that with a Docker container based setup, it's possible to attain a level of operational robustness that exceeds the typical requirements of distributed data-intensive businesses.

10 Lessons learned

- Increasing the `nofile` limit to very large values (e.g. 512M) exposes some undesirable behavior in tools such as `yum/rpm` that attempt to close all file descriptors. This led to excessive CPU usage and very long delays for any workflow involving these utilities, including Ansible.
- Having an easy-to-use REST API client library and documented service API goes a long way to making integration and testing simple tasks.
- The decision to enable or disable hyperthreading can actually be made during the disk I/O benchmarking stage. If enabled hyperthreading helps yield higher storage throughput statistically, then keep it. Otherwise, leave it alone or disable it.
- Using a modern disk I/O benchmarking tool such as `elbencho` is really helpful to any serious data movement endeavors.
- With a well-tuned infrastructure, it is feasible for a software data mover to use a single setting for a wide range of file sizes and various size distributions (i.e. file size histograms).
- More focused tuning may improve the results for a specific file size range or distribution, but the return may not justify the effort.
- Even tuning `zx` takes the adjustments of only a few parameters and is straightforward, the results from properly and insufficiently tuned `zx` can be very signficiant (e.g. ≥ 200%). Below is a real example using the aforementioned NERSC production dataset[36]

`K`	**`piece_size`** (MiB)	Mean speed over 100G SDN testbed (Gbps)
16	8	30
24	32	68

- Some challenges encountered while attempting to use `fio`:
    a. If a dataset has LOSF, `fio` doesn't allow the use of the preferred block size of the application. In addition, when `fio` is configured using `thread`ing and many `numjobs`, even with a large enough number of open files and with the `fio --alloc-size` option set to a very large number of KBs, `fio` core dumps frequently.
    b. Furthermore, If a dataset is of a mix-sized type, but predominantly LOSF, `fio` may "run", but under reports the attainable storage throughput.

[36] Private email exchanges between Ezra Kissel and the Zettar team, October 27, 2020.

c. The numerous `fio` options on the one hand provide a high-degree of flexibility, on the other hand take tremendous effort to master. It's extensive documentation [37] is indirect proof.

11 Conclusions

- Properly provisioned, configured, and for data rates ≤ 100Gbps, a containerized environment has a high potential to meet the project goals: *to provide optimized, on-demand data movement tools/endpoints to users of ESnet*.
- When modern TCP is used efficiently by a data movement service to move data at scale and speed, network latency becomes less of a factor - the same level of data rates are attainable over LAN, Metro, and WAN as long as loss rates remain low.
- For really optimal results, the storage, the host, and the network should always be tuned first. This may need a few iterations - *for example, as the host is tuned, the attainable storage throughput observed in the host may increase*. *The examples shown in the* ***Appendix*** *should be regarded as starting points*.
- Data movements always involve at least a data source (reading) and a data target (writing). As such, *storage devices of high write performance are essential to avoid bottlenecks*. Note that common flash storage devices almost always provide higher read performance than write, with Intel Optane SSDs[38] being a notable exception.
- Using a containerized environment definitely facilitates on-demand provisioning of DTNs. Nevertheless, note that such approaches may improve the utilization of computing resources such as CPUs. But with efficient, I/O intensive applications, the use of multiple containers running `zx` per host may yield little benefit, if any. However, `zx` has the potential to be used in a multi-tenant mode of operation in conjunction with other containerized transfer endpoints. Such a scenario may require scheduling and/or bounding network and storage I/O rates depending on the hardware availability and sharing model.
- Furthermore, for point-to-point data rates in the multiple 100Gbps - Tbps range, e.g. for the Linac Coherent Light Source II[39], an efficient and intrinsically scale-out[40] software data mover and the use of a distributed HPC storage (both file and object) become mandatory. How a container-based approach fits this use case still needs to be evaluated in a follow-up work.

---

[37] "Welcome to FIO's documentation!". https://fio.readthedocs.io/en/latest/. Accessed Nov. 26, 2020.
[38] "Intel® Optane™ DC SSD Series". https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/optane-dc-ssd-series.html. Accessed 21 Nov. 2020.
[39] "LCLS-II Design and Performance". https://lcls.slac.stanford.edu/lcls-ii/design-and-performance. Accessed 20 Nov. 2020.
[40] That is to scale out like e.g. the Lustre parallel file system, without the use of a cluster workload manager or a proprietary orchestration application.

12 Acknowledgements

We gratefully acknowledge the support of ESnet, in particular that of Inder Monga and Chin Guok for their support and encouragement. Les Cottrell and Wilko Kroeger, SLAC National Accelerator Laboratory, for their careful reviews of the draft and helpful comments. Sven Breuner, Excelero, for his excellent `elbencho` distributed storage benchmark for file systems and block devices. The timely appearance of this tool has made the storage benchmarking effort of this project much more tractable. Andrey O Kudryavtsev, Intel NSG, for his valuable comments and insights about NVMe SSD based HPC storage. We also wish to give credit to the excellent software engineering of Igor Soloviov and Oleksandr Nazarenko, both of Zettar Inc., for the robust and efficient Zettar `zx` implementation.

ESnet is funded by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research. Benjamin Brown is the ESnet Program Manager.

ESnet is operated by Lawrence Berkeley National Laboratory, which is operated by the University of California for the U.S. Department of Energy under contract DE-AC02-05CH11231.

Appendix

A.1 Sysctl settings applied on each host:

```
{
    "kernel.pid_max": "4194303",
    "net.ipv4.tcp_max_syn_backlog": "4096",
    "net.ipv4.tcp_fin_timeout": "15",
    "net.ipv4.tcp_wmem": [
        "4096",
        "87380",
        "536870912"
    ],
    "net.ipv4.tcp_rmem": [
        "4096",
        "87380",
        "536870912"
    ],
    "net.ipv4.udp_rmem_min": "16384",
    "net.ipv4.tcp_window_scaling": "1",
    "net.ipv4.tcp_slow_start_after_idle": "0",
    "net.ipv4.tcp_timestamps": "1",
    "net.ipv4.tcp_low_latency": "1",
```

```
    "net.ipv4.tcp_keepalive_intvl": "15",
    "net.ipv4.tcp_rfc1337": "1",
    "net.ipv4.tcp_keepalive_time": "300",
    "net.ipv4.tcp_keepalive_probes": "5",
    "net.ipv4.tcp_sack": "1",
    "net.ipv4.neigh.default.unres_qlen": "6",
    "net.ipv4.neigh.default.proxy_qlen": "96",
    "net.ipv4.ipfrag_low_thresh": "446464",
    "net.ipv4.ipfrag_high_thresh": "512000",
    "net.core.netdev_max_backlog": "250000",
    "net.core.rmem_default": "16777216",
    "net.core.wmem_default": "16777216",
    "net.core.rmem_max": "536870912",
    "net.core.wmem_max": "536870912",
    "net.core.optmem_max": "40960",
    "net.core.dev_weight": "128",
    "net.core.somaxconn": "1024",
    "net.ipv4.udp_wmem_min": "16384",
    "net.core.default_qdisc": "fq_codel"
}
```

A.2 Dockerfile.systemd:

```
FROM centos/systemd

RUN yum update -y
RUN yum install -y wget vim net-tools iperf3 python3 openssh-server
openssh-clients sudo initscripts
RUN yum clean all

ENV SSH_PORT 4023

# iperf3
EXPOSE 5201

RUN sed -ri "s/^#?Port\s+.*/Port $SSH_PORT/" /etc/ssh/sshd_config
RUN sed -ri 's/^#?PermitRootLogin\s+.*/PermitRootLogin yes/'
/etc/ssh/sshd_config
RUN sed -ri 's/UsePAM yes/#UsePAM yes/g' /etc/ssh/sshd_config
RUN sed -ri 's/# %wheel/%wheel/g' /etc/sudoers

#RUN mkdir /root/.ssh
RUN ssh-keygen -A

CMD [ "/sbin/init" ]
```

A.3 `fio` job file

```
[writetest]
thread=1
blocksize=16m
rw=randwrite
direct=1
buffered=0
ioengine=psync
gtod_reduce=1
numjobs=12
iodepth=1
runtime=180
group_reporting=1
percentage_random=90
opendir=./gridftp
```

A.4 Histogram of the ESnet sample production dataset

Order	Number	Accumulated size
0	1	0
512	626	611.33 KiB
1. KiB	626	1.79 MiB
4. KiB	626	4.78 MiB
8. KiB	1	4.79 MiB
16. KiB	1	4.8 MiB
32. KiB	1	4.85 MiB
64. KiB	1	4.95 MiB
128. KiB	1	5.14 MiB
256. KiB	1	5.62 MiB
512. KiB	627	603.57 MiB
1. MiB	626	1.76 GiB
4. MiB	626	4.67 GiB
8. MiB	747	11.63 GiB
16. MiB	716	24.96 GiB
32. MiB	627	54.16 GiB
64. MiB	627	112.55 GiB
128. MiB	626	229.16 GiB
256. MiB	626	520.66 GiB
512. MiB	2	522.52 GiB
1. GiB	141	693.65 GiB
2. GiB	27	786.91 GiB
4. GiB	12	853.58 GiB

```
8. GiB              2                   872.2 GiB
16. GiB             12                  1.09 TiB
32. GiB             2                   1.18 TiB
64. GiB             4                   1.59 TiB
256. GiB            1                   2.04 TiB
------------------------------------------------------------
Total files: 7936
Total size: 2.04 TiB
Average file size: 269.92 MiB
```

A.5 `zx` settings[41]

Use case	K	Piece_size (MiB)	downloader	uploader	fs	net

The generic applicable zx settings	64	64	`downloading_piece_num = 96`
			`chunk_size = 16MiB`
			`file_direct = true`
			`aio = false`
			`truncase = true`
				`file_direct = true`
				`aio = false`
					56	24

1PB TLS transfer	56	64	`downloading_piece_num = 96`
			`chunk_size = 16MiB`
			`file_direct = true`
			`aio = false`
			`truncase = true`
				`file_direct = true`
				`aio = false`
					24	24

1PB non-TLS transfer[42]	24	32	`downloading_piece_num = 96`
			`chunk_size = 16MiB`
			`file_direct = true`
			`aio = false`
			`truncase = true`
				`file_direct = true`
				`aio = false`
					12	32

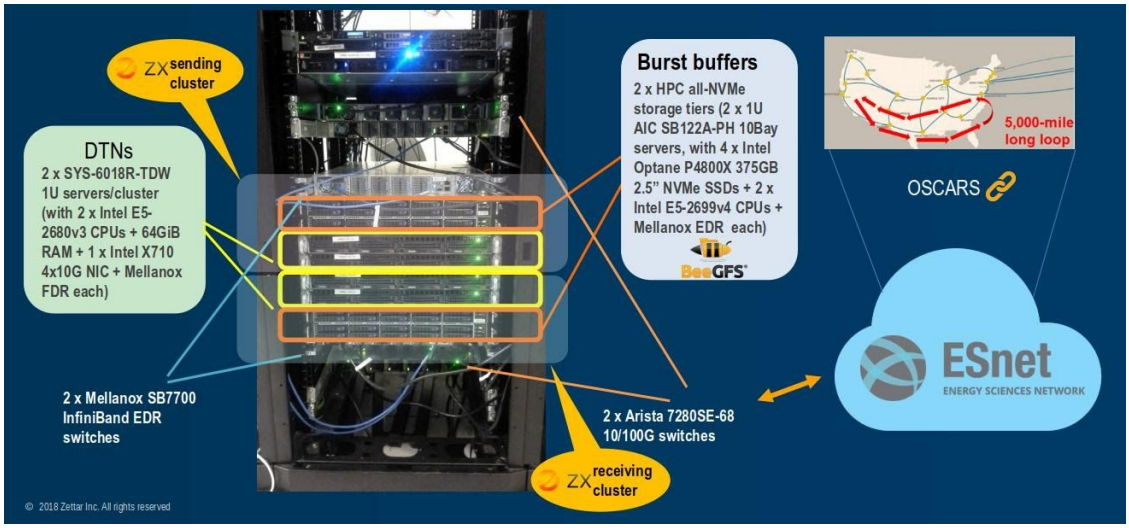
A.6 Possible causes of the low TLS transfer rates

During the SLAC/Zettar high-speed transfer research, a production trial run[43] carried out on September 27, 2018 achieved 94% of the capped bandwidth, 80Gbps. The run also employed full TLS encryption, unconditional check summing, and a 1PB simulated production dataset. Comparing the two setups, some obvious differences are described below:

[41] All these options are described in the `zx` online documentation and man page (access controlled).
[42] In the [ddc] section of the zx configuration file, zx.conf, **`task_progress_timeout = 300`**
[43] "ESnet's Network, Software Help SLAC Researchers in Record-Setting Transfer of 1 Petabyte of Data". https://www.es.net/news-and-publications/esnet-news/2018/esnets-network-software-help-slac-researchers-in-record-setting-transfer-of-1-petabyte-of-data/. Accessed Nov. 26, 2020.

A.6.1 The SLAC/Zettar data transfer setup

The picture below shows the setup used for the aforementioned production trial run. It used an ESnet OSCARS[44] 100Gbps, 5000 miles long loop. So, both ends actually were on the same rack. This topology is similar to what this project employs.



- The above is a scale-out setup. At each end, instead of a single server, there are two servers, albeit far less powerful models and older (3 years old at the time of the trial). Each server ran a `zx` instance, which is intrinsically cluster-oriented - it doesn't need a cluster workload manager or an orchestration application.
- Each server has an Intel X710[45] 4x10Gbps NIC. So, each port's speed is 1/10th of the current setup's Mellanox ConnectX5's port speed.
- The 4 ports of each X710 NIC were not bonded. Instead, `zx`'s built-in ability to aggregate multiple network interfaces presented in the OS was utilized.
- Even the setup shown above used BeeGFS[46], a parallel file system, but the storage devices employed, Intel Optane SSDs, provide far better write performance than the current Western Digital Ultrastar SN200, at low queue length.
- Two Arista 7280SE-68[47] 10/100G switches were used.
- The setup was located in SLAC's Science DMZ, but all the DTNs were configured with
    - `selinux`[48] disabled - Zettar's view is that complexities actually promotes vulnerability

---

[44] "OSCARS". https://www.es.net/engineering-services/oscars/. Accessed Nov. 26, 2020.
[45] "Intel® Ethernet Converged Network Adapter X710-DA2/DA4". https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-x710-brief.pdf. Accessed Nov. 26, 2020.
[46] "BeeGFS, The leading parallel File system". https://www.beegfs.io/c/. Accessed Nov. 26, 2020.
[47] "Arista Quick Start Guide 7280 Series 1 RU (Gen 3) Data Center Switches". https://www.arista.com/assets/data/pdf/qsg/qsg-books/QS_7280_1RU_Gen3.pdf. Accessed Nov. 26, 2020.
[48] "SELINUX USER'S AND ADMINISTRATOR'S GUIDE". https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index. Accessed Nov. 30, 2020.

- `firewalld`[49] disabled - Zettar's approach is to ensure the configuration is done right and to install a minimal set of software - A cracker cannot cause damage with something not on the system. DTNs do not allow end user login either. Users stage their data in the storage pools that are imported into the DTNs.

The above setup spreads the higher TLS computational workloads to more hardware components, even though each one is of modest spec.

A.6.2 Sysctl tuning

In Zettar's experience gained during the SLAC/Zettar high-speed data transfer research, `zx` doesn't need very aggressive `sysctl` and NIC tuning. For `sysctl` tuning, only the following were introduced:

```
net.core.rmem_max = 25165824
net.core.wmem_max = 25165824
net.ipv4.tcp_rmem = 4096 87380 25165824
net.ipv4.tcp_wmem = 4096 65536 25165824
net.ipv4.tcp_mtu_probing=1
net.core.somaxconn=2048
```

In addition, `tuned`[50] profile for each host was set to `throughput-performance`. CPU frequency was set at 2.5Ghz constant. The fan speed was adjusted to keep the CPU temperature always below the critical level to avoid CPU throttling.

A.6.3 NIC tuning

```
/sbin/ethtool -A $1 rx off tx off
/sbin/ethtool -G $1 rx 4096 tx 4096
/sbin/ethtool -K $1 tx off sg off tso off
/usr/sbin/ip link set $1 txqueuelen 10000
```

We will further investigate the impact of such hardware and tuning differences in follow-up work.

---

[49] "5.3. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD". https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-viewing_current_status_and_settings_of_firewalld. Accessed Nov. 30, 2020.

[50] "3.2. PERFORMANCE TUNING WITH TUNED AND TUNED-ADM". https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/performance_tuning_guide/sect-red_hat_enterprise_linux-performance_tuning_guide-performance_monitoring_tools-tuned_and_tuned_adm. Accessed Nov. 26, 2020.