Benchmarking NetBASILISK: a Network Security Project for Science

Jem Guhit[1], *Edward* Colone[2], *Shawn* McKee[1], *Kris* Steinhoff[2,3], *Katarina* Thomas[2]

[1]Physics Department, University of Michigan, Ann Arbor, MI, USA
[2]Information and Technology Services, University of Michigan, Ann Arbor, MI, USA
[3]School of Information, University of Michigan, Ann Arbor, MI, USA

Abstract. Infrastructures supporting distributed scientific collaborations must address competing goals in both providing high performance access to resources while simultaneously securing the infrastructure against security threats. The NetBASILISK project is attempting to improve the security of such infrastructures while not adversely impacting their performance. This paper will present our work to create a benchmark and monitoring infrastructure that allows us to test for any degradation in transferring data into a NetBASILISK protected site.

1 Introduction

Distributed computing infrastructures face significant challenges in effectively supporting scientists in moving, accessing, analyzing and transforming data to produce new scientific insights. This is complicated by continuous attacks that all systems connected to the Internet experience[1][2]. The challenge is to secure the infrastructure without compromising the performance and usability of the infrastructure.

The Energy Sciences network (ESnet)[3] created a new paradigm in 2010 to help address this situation: the Science DMZ [4]. A network DMZ, or demilitarized zone, is a physical or logical sub-network that contains and exposes an organization's external-facing services to an un-trusted network such as the Internet. The DMZ functions as a small, isolated network positioned between the Internet and the private network [5]. This paradigm allowed data transfer nodes to bypass firewalls and security devices which would otherwise interfere with science data-flows, adversely impacting scientists in their work. The intent is to minimize the number of devices attached to the Science DMZ and carefully monitor and configure them to provide both performance and security.

The NetBASILISK [6] (NETwork Border At Scale Integrating and Leveraging Individual Security Components) project, led by researchers and network engineers at the University of Michigan, seeks to augment and expand the Science DMZ concept. The goal is to allow institutions to maintain both security and capability for all their users by prototyping and deploying a network border security solution capable of supporting unfettered network traffic at 4x100 Gbps using a mix of commercial offerings and locally-developed middleware.

In this paper we will describe the work we have done to evaluate the effectiveness of the NetBASILISK infrastructure and verify that it does not adversely impact the ability of the ATLAS scientists at the University of Michigan to move data from outside our institution.

2 Rationale and Project Components

A network security infrastructure has the potential to impact data entering or leaving an institution. The focus of the work described in this paper is evaluating the impact of NetBASILISK on data coming into the University of Michigan.

To understand if NetBASILISK impacts the transfer of external data to the university, we need to:

- Determine normal transfer behavior in the absence of NetBASILISK (get a baseline).
- Monitor relevant factors **other** than NetBASILISK that might impact transfers.
- Create a benchmark test which can be run on-demand to determine transfer rates.

Using this set of information and tools, we can evaluate if NetBASILISK (or changes in NetBASILISK) are adversely impacting our ability to transfer data to the University of Michigan. The two capabilities we need to evaluate NetBASILISK's potential impact are a **benchmark application**, which can be used both to derive a baseline and make on-demand evaluation of the current transfer performance, and an **environmental monitoring application** which can gather and track relevant metrics that might be related to the observed transfer performance. In the following section we will describe the development of the benchmark and, later, the environmental monitoring application development.

3 Development of the Benchmark

The NetBASILISK benchmark application aims to build a stable transfer file system that would derive a baseline for normal transfer behavior and evaluate transfer performance. There are two main components to the benchmark application: **data** and **file transfer tool**.

3.1 Data

Since the University of Michigan Physics department is a collaborator of the ATLAS Experiment [7], the data would be coming from a repository containing samples of ATLAS datasets located at Brookhaven National Laboratory (BNL). BNL is an ideal choice for an external data source because it is one of the primary collaborators for data operations, it has a well-defined network for data transfers coming into the University of Michigan, and it already has a data repository intended to be accessible for long-term that could be used for the benchmark test.

3.2 Transfer Tool

After securing an external data source, a software framework tool for grid computing is used to access the ATLAS data repository and transfer files from BNL to the University of Michigan. Various software tools exist to handle file transfers. We decided to choose from the five common transfer tools used in the High Energy Physics Computing toolkit: File Transfer Service (FTS) [8], Storage Resource Management (SRM) [9], Grid File Access Library (GFAL) [10], Globus GridFTP [11], and XRootD [12].

To ensure that the benchmark test is reliable and consistent after successive test runs, we imposed a criteria that the transfer tool has to satisfy, such as:

(a) Stable Transfer Speed
(b) Consistency
(c) Checksum Verification
(d) Information about the destination storage nodes

3.2.1 Transfer Tool Evaluation

The transfer tool's capabilities are tested by running multiple data transfers using a subset from the ATLAS data repository, specifically transferring the same ten datasets ten times. Originally we planned to run the test only once, however, two transfer tools satisfied the criteria shown in Table 1, thus another round of the test had to be done. At the end, Transfer Tool Evaluation evolved to encompass two test rounds: the first focused on whether the storage clients satisfy the criteria, second tested the variability and consistency of the storage clients that passed the initial criteria.

Storage Client	**Criteria**			
	a. Transfer Speed	b. Consistency	c. Checksum Verification	d. Storage Node
FTS	✗	✗	✓	✗
SRM	✗	✓	✓	✗
GFAL	✓	✓	✓	✗
Globus	✓	✓	✓	✓
XRootD	✓	✓	✓	✓

Table 1: Results of the first round of the storage client test. The table shows the requirements satisfied by each storage client.

Table 1 summarizes the first test round: it shows how different transfer tools satisfy the established criteria . From the first round, three storage clients were ruled out: FTS, SRM, and GFAL. FTS did not satisfy criteria (a), (b), and (d): its non-trivial way to extract the transfer rate and destination storage node information from the log file failed criteria (a) and (d), FTS' own system for scheduling data transfers would limit our flexibility to control data transfers thus it failed criterion (b). SRM did not satisfy criteria (a) and (d): for (a), the protocol revealed a huge variability in transfer rates, while for criterion (d) it had difficulties in extracting the destination storage node information from the log files. GFAL did not satisfy criterion (d), because it did not provide information about the destination storage node. Globus and XRootD both satisfied all the criteria and went to the next round of testing.

The second test iteration, although virtually same as the round one, focused on finding out which storage client could provide the most stable file transfer performance over a period of time. In addition, information such as the average and standard deviation for the transfer rates was gathered for comparison. Figure 1 shows that after the second round of test between Globus and XRootD, the average and standard deviation of the former had shown higher transfer rate variability. This determined that XRootD protocol is better suited to use as the official transfer tool for the benchmark application.

4 Environmental Monitoring Application

Having a benchmark to evaluate transfer performance is critical for our task but it is equally important to understand the environment in which the benchmark is run. A poor transfer result could be because NetBASILISK has adversely impacted the transfer **OR** it may be due to a number of other possibilities:

- The network between the benchmark source data and destination may be congested at the source or destination local area network, or in the wide area network.
- The source data servers might be overloaded or have other resource issues.
- The destination data servers might be overloaded or have other resource issues.

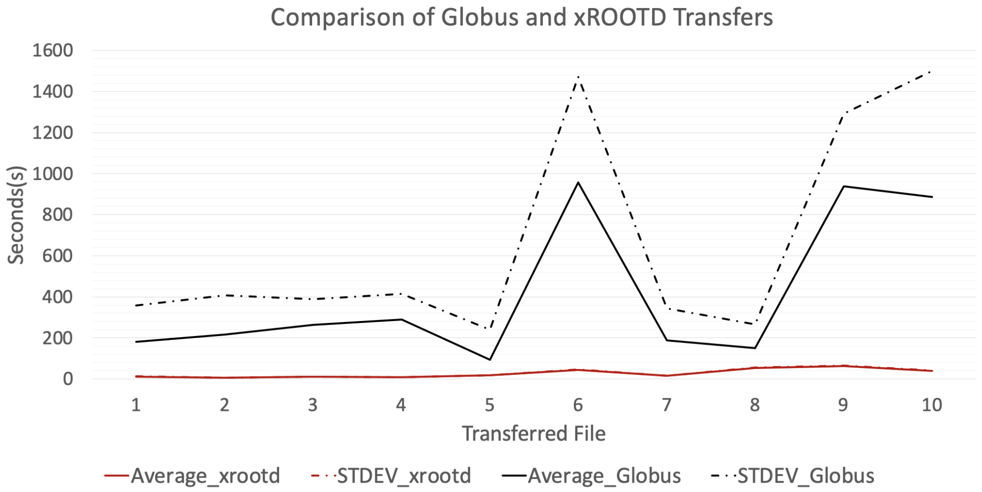


Figure 1: The tests transferred the same ten files ten times for each tool. The STDEV and Average for the xROOTD case are almost overlapping. The figure shows that Globus has more fluctuation and instability compared to xROOTD. The total time for 10 tests for Globus is 11.5 hours while total time for 10 tests in xROOTD is 3.79 hours.

To determine the cause of a poor benchmark result we have to gather sufficient information about the environmental conditions under which the benchmark was executed. Thus we need information on the end systems and the network in between, including the Brookhaven National Laboratory (BNL) storage systems (source of the benchmark data); the network path between BNL and the ATLAS Great Lakes Tier-2 [13] cluster at the University of Michigan; and the storage nodes at AGLT2 (destination for the benchmark data).

4.1 Network Monitoring

To monitor the network status, we needed to identify the existing network monitoring mechanisms along our network path. For the environment monitoring script, the path segments were defined as the source network at BNL, the WAN between our sites and the destination network at AGLT2. Although we considered measuring the round-trip time (RTT), we found that it was not as relevant for our use-case.

Network at Source: While network monitoring internal to BNL exists, we didn't have easy access to those systems. The internal networking at BNL is well provisioned and we are assuming that the BNL network will not limit our benchmark transfers. As we continue to evolve the environment monitoring, we intend to include explicit BNL network monitoring.

Wide-area Network: The wide area network path between BNL and AGLT2 has two distinct components: 1) a path from BNL to Chicago on ESnet, and 2) a path from ESnet's location in Chicago to AGLT2. We will discuss each components below. For the 1) BNL to Chicago component we encountered some challenges in monitoring the network path. ESnet, which handles the part of the path, provides publicly accessible monitoring information[14]. Figure 2 shows the ESnet traffic monitoring, including the possible paths between BNL to Chicago (CHIC) from the my.es.net website. Unfortunately, because ESnet uses MPLS[15] to control traffic between their ingress and egress points, we don't have a way to identify **which** interfaces and paths our benchmark traffic traversed. In our current environment monitoring, we assume the ESnet portion of the path will not be congested and not

https://doi.org/10.1051/epjconf/202125102068

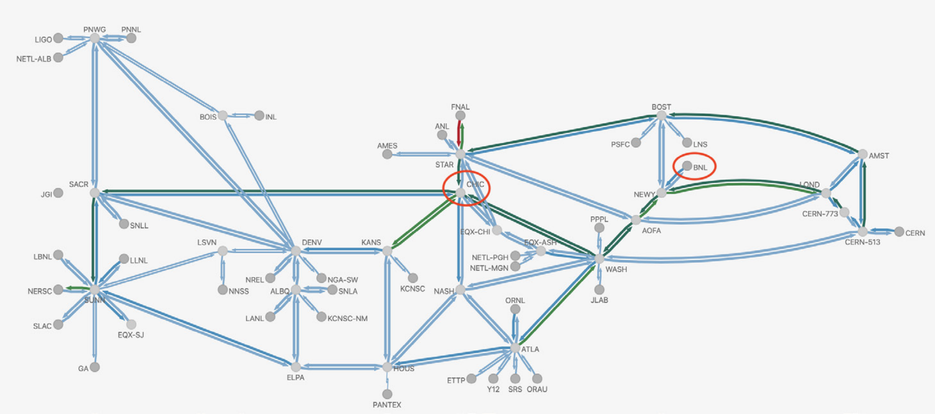


Figure 2: The ESnet network monitored on their portal. Shown are multiple paths from BNL to CHIC (Chicago), emphasized in red circles, where our benchmark traffic would exit ESnet to connect to AGLT2. ESnet uses internal traffic engineering which precludes us from knowing which of the many possible physical paths between BNL and CHIC are followed for a given transfer.

contribute to a poor benchmark result due to their traffic engineering. We have been in contact with the ESnet developers and will try to include explicit monitoring in a future update.

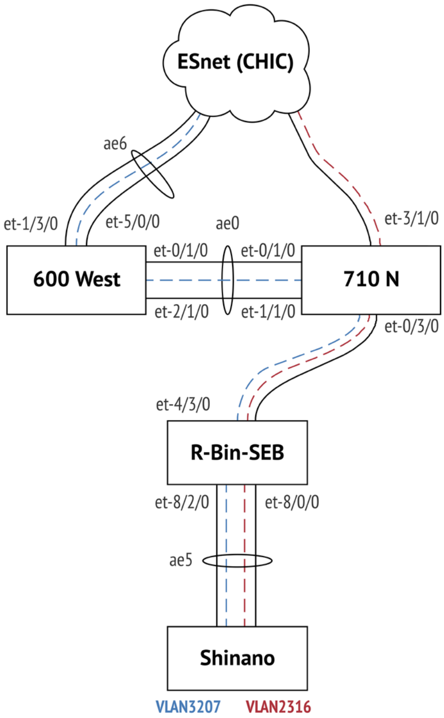


Figure 3: Network Topology between Shinano (AGLT2) and ESnet (CHIC) showing the internal and external routers, ports, port-chanels, and VLAN line for each server.

The second part of the WAN path, Chicago to AGLT2, has more useful monitoring available. We monitor and verify that all the interface along our path are not congested. In Figure 3 we can show the routers and interfaces involved, starting from the ESnet presence in Chicago (at the top of the figure) and connecting all the way to the AGLT2 border router (named "Shinano") at the bottom. The environment metrics of the Shinano Router (and AGLT2 LAN) are available from the CheckMK interface [16] and is discussed in more detail in Section 4.2. R-Bin-SEB is an internal router for the University of Michigan and monitored by the CA Performance Center. We gather the following metrics using a python script developed to access the Application Programming Interface (API) of the CA Performance Center [17]:

- Bits In/Out: converted to GBs In/Out
- Bits/s In/Out: converted to GB/s In/Out
- Utilization In/Out: % of capacity
- Error In/Out: number of packets that caused an error (e.g failed checksum)

Buffer Overflow Discards, where packets are dropped when the buffer on either the incoming or outgoing port is full, is another metric being considered for R-Bin-SEB and will be included if accessible. The routers in Chicago (710 N / 600 West) have metrics stored in Grafana [18]. We gather the following metrics from a python script developed to interact with Grafana [19]:

- Input: measured in GBps
- Output: measured in GBps

4.2 End-system Monitoring

In addition to the network we need to understand the operating conditions of the end-systems at the source and destination sites.

BNL End-System: As mentioned in Section 4.1, even though a monitoring interface exists to capture metrics from BNL, accessing the system is currently non-trivial. In the future update, we plan to include the environment metrics for BNL and the script to access the API interface.

AGLT2 End-System: As mentioned in Section 4.1, the IT Infrastructure monitoring software for the AGLT2 Cluster System is Check MK [16]. All transferred files sent to the AGLT2 Cluster are randomly stored in one of the dCache [20] servers. Check MK contains a variety of network metadata of each dCache server and the metrics of interest for the benchmark script would be the the following:

- **CPU Load**: number of active processes in the system.
- **CPU Utilization**: estimates system performance in percentage.
- **Memory**: variables of interest are MemFree and MemAvailable measured in GB.
- **Disk Input Output (IO)**: contains the read and write operations of the physical disk and is usually measured in GBps. The variable of interest for us is the Disk Utilization measured in percentages.

A python script was developed to gather these metrics from the Application Programming Interface (API) of the CheckMK Interface[21].

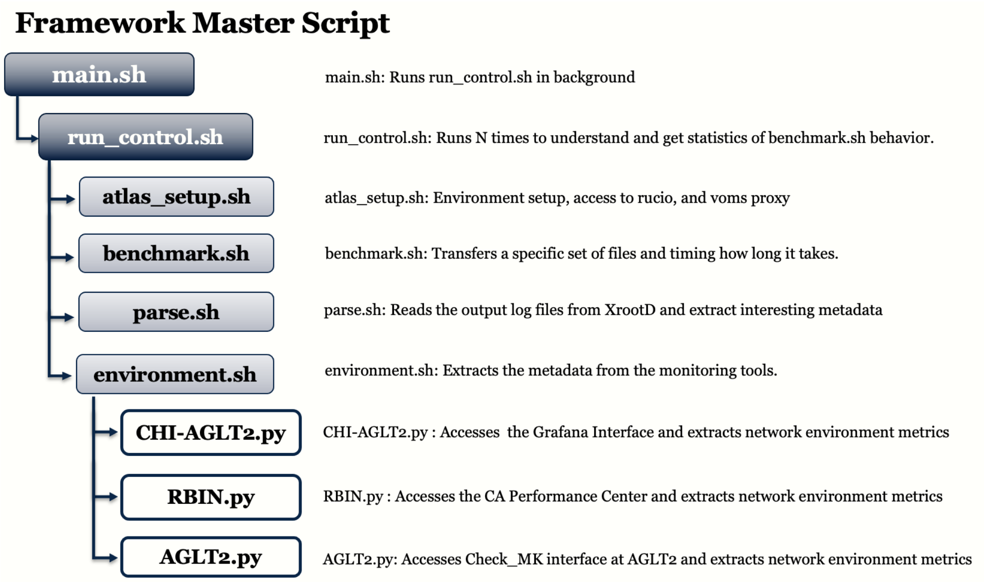
The environment metrics described in this section are important components that would allow us to monitor the network status along the transfer path. The following section will describe how the benchmark application and environment monitoring application come together as a whole framework.

5 Framework Implementation

The benchmark application discussed in Section 3 and the environment monitoring application in Section 4 together are the backbone of the framework composed of shell and python scripts shown in Figure 4. It shows the first version of the framework script where it runs the benchmark application followed by the environment monitoring application.

The `main.sh` script holds the framework together and runs the scripts in background to ensure running processes do not get aborted automatically when the SSH connection is lost. It contains a sub-module `run_control.sh`, which runs four sub-modules. The `atlas_setup.sh` sets up the XRootD software package. The `benchmark.sh` script is the beginning of the benchmark application component and handles the file transfers between BNL and AGLT2. This script also records the `Start` and `End` times of each benchmark test to be used for the `environment.sh` module. Each benchmark test produces a log file which contains important information about the transfer, more on Section 5.1 which are extracted in `parse.sh` and organizes the output metadata into json files. This finishes the benchmark application portion and we move on to the environment monitoring application component.

As mentioned earlier, the `Start` and `End` times recorded from `benchmark.sh` are used as input variables for the three sub-modules of the `environment.sh`. We initially implemented that the environment monitoring application captures the network metrics in the same time-range the benchmark test was ran. The environment monitoring component has three independent scripts for the AGLT2 End-System (`AGLT2.py`) and Wide Area Network path for Chicago to AGLT2 (`CHI-AGLT2.py`, `RBIN.py`). These scripts have a similar structure of capturing network metrics but had to be scripted independently because of different API syntax of the network interfaces.



main.sh: Runs run_control.sh in background

run_control.sh: Runs N times to understand and get statistics of benchmark.sh behavior.

atlas_setup.sh: Environment setup, access to rucio, and voms proxy

benchmark.sh: Transfers a specific set of files and timing how long it takes.

parse.sh: Reads the output log files from XrootD and extract interesting metadata

environment.sh: Extracts the metadata from the monitoring tools.

CHI-AGLT2.py : Accesses the Grafana Interface and extracts network environment metrics

RBIN.py : Accesses the CA Performance Center and extracts network environment metrics

AGLT2.py: Accesses Check_MK interface at AGLT2 and extracts network environment metrics

Figure 4: First version of the framework master script where it runs the benchmark application followed by the environment monitoring application.

The first version included the environment monitoring application in the framework script. However, we wanted to track the variability of the environment monitoring application in the long-term and not only when the benchmark application was ran. Therefore, we decided to detach the environment monitoring script from the benchmark application script and make it its own continuous operating service.

5.1 Benchmark Application Metrics

As described above, a bash script `parse.sh` extracts the following metadata from the benchmark file transfers:

- **Bandwidth**: The average rate of the benchmark transfers measured in MB/s.
- **Transfer Time**: Total time to transfer all files measured in seconds.
- **Destination Host Frequency**: Count of files sent to each destination host.
- **File Size**: The size of each file transferred measured in GB. Summing gives us the total.
- **Transfer Speed**: The transfer speed of each file measured in MB/s.

These benchmark metrics are stored in a json file on the execution host and sent to Humio [22] for plotting and analysis. The benchmark results correspond to a benchmark score determining whether the average transfer speed of the test is **OK**, **WARNING**, or **CRITICAL**.

5.2 Environment Monitoring Application Metrics

As described above, a bash script `environment.sh` runs the three sub-modules: `CHI-AGLT2.py, RBIN.py, and AGLT2.py` to extract the environment metadata discussed in Section 4. `CHI-AGLT2.py` extracts the metadata from routers in Chicago (710N /600 W) stored in the Grafana interface and saves it in json format. Similarly `RBIN.py` extracts metadata from the R-Bin-SEB router stored in the CA Performance Center. A pre-processing script [23] takes the json outputs and calculates important statistics (mean, standard deviation, min, and max). These results correspond to a benchmark score determining whether the network activity in the Chicago and AGLT2 path is **OK**, **WARNING**, or **CRITICAL**.

As for the AGLT2 End-System, `AGLT2.py` extracts the benchmark metadata defined in Section 5.1 for each dCache server from the Check MK interface and saves it in json format. A pre-processing script [23] takes the json output and calculates the statistics (mean, standard deviation, min, and max) for each dCache server. Since the transferred files are randomly assigned to the dCache servers, the **destination of host frequency** variable defined in Section 5.1 is used to normalize and re-weight the statistics. These results correspond to the benchmark score determining whether the AGLT2 network activity at a specific time frame is **OK**, **WARNING**, or **CRITICAL**.

6 Interpreting Results

This section describes how we interpret the results coming from the benchmark application and environment monitoring application components.

6.1 Framework Score

The framework score is based on the results of the **(1)Benchmark Application** which provides statistics from the file transfers from BNL to AGLT2. Then **(2) Environment Monitoring Application** which queries network metadata from the **(2)a Source**, **(2)b Network In Between**, and **(2)c Destination**.

Three threshold levels are defined for each component: **OK**, **WARNING**, and **CRITICAL**. **OK** means fast transfer time in the context of (1) and a congestion free network in the context of (2)a - (2)c. **WARNING** means medium transfer time in the context of (1) and a medium-congested network in the context of (2)a - (2)c. **CRITICAL** means slow transfer time in the context of (1) and a congested network in the context of (2)a - (2)c. The threshold level for (2) is determined by the worst result of (2)a - 2(c).

6.2 Framework Score Example

How is the Benchmark Score interpreted to provide information about NetBASILISK? This is best explained by two cases presented in Table 3. Case 1 shows that the cause of a **CRITICAL** transfer time might be due to NetBASILISK. The Testing Environment is designated as **OK**, meaning the environment metrics indicate a congestion free network. However, the transfer time is slow which means that NetBASILISK might have played a role in this performance degradation. Case 2 indicates that even if the time transfer is **CRITICAL**, the slow transfer time might not be caused by NetBASILISK as the Testing Environment is **CRITICAL** also: the environment metrics indicate a congested network which might have been a cause for the slow transfer time.

7 perfSONAR Testing

In order to correlate data transfer benchmarks with prevailing network conditions, NetBASILISK uses perfSONAR [24] to run periodic latency and bandwidth tests between the relevant network locations. perfSONAR is an open-source network metric gathering toolkit

Framework Components	**Case 1**	**Case 2**
Source	OK	CRITICAL
Network In Between	OK	WARNING
Destination	OK	OK
Testing Environment	OK	CRITICAL
Ave Transfer Speed	CRITICAL	CRITICAL

Table 2: In Case 1, all environment metrics are OK but poor transfer results are observed, indicating a likely problem caused by NetBASILISK. Case 2 shows an example where the Source is the likely cause of the poor transfer result.

that allows for ad-hoc and scheduled testing between perfSONAR testpoints. It uses existing, standard tools such as iperf3 and owamp to generate performance metrics.

The benefit of using perfSONAR is to generate network performance metrics which can be contrasted against application- and service-level metrics. In theory, if we observe poor service metrics, we can determine if the network performance has impacted the service by using perfSONAR to generate throughput and latency measurements around the time of the substandard performance. If the network metrics are within the established performance thresholds, we can infer that other infrastructure related to the service, such as CPU, disk, relative load, and other conditions not related to the network are the cause of the poor service performance. If the network performance metrics are below the threshold, we can assert that the network itself is partially or wholly the cause of poor service performance.

To gauge NetBASILISK's ipact on performance, we will run scheduled metric gathering tests. This testing activity will alternate between simulated science activities interspersed with perfSONAR throughput and latency measurements. As AGLT2 represents data ingress, we will run file transfers from BNL to the University of Michigan. We will also utilize publicly available perfSONAR nodes to test throughput and latency from BNL to the University of Michigan to gauge metrics before and after the file transfers. This scheduled activity is run with cron at AGLT2. The script that is executed is an Ansible playbook [25] that alternates perfSONAR testing with file transfer test activities. The Ansible playbook has provisions for interacting with pScheduler, and the benchmark scripts.

8 Conclusion and Future Work

To relate test result data with other science drivers, the NetBASILISK project will explore the use of the Humio log platform [22] to aggregate test results from a variety of benchmark and perfSONAR tests.

We plan to update our benchmark implementation to automatically send results to the Humio API to include those test results in the repository shared with other parts of the NetBASILISK project. This work will involve evaluating the best ways to represent the benchmark outcome data in a way that it can be effectively analyzed.

The perfSONAR project is transitioning to the Elastic Stack platform [26] for data archiving and visualization. The perfSONAR team has published Logstash pipelines in order to facilitate data analysis of perfSONAR results [27]. To this end we are standing up an Elastic Logstash server and using the perfSONAR pipelines. Humio is able to ingest data with the same methods as Elastic, so we are going to use Logstash to import the data to Humio.

The ultimate goal is to be able to combine similar benchmark and network test results into a single repository. This will allow analysis across the variations and could ultimately provide for more detailed monitoring and alerting. We hope to use Humio's capabilities as a data visualizer to contrast science driver results with perfSONAR and other metrics.

9 Acknowledgements

We gratefully acknowledge the National Science Foundation which supported this work through NSF grant OAC-1925476. In addition, we acknowledge contributions from the rest of the NetBASILISK team.

Process	LepHad SLT SR (%)	LepHad LTT SR (%)	Description
VBF	0.863	0.624	Scales
VBF	0.149	0.233	PDF+α_s
VBF	6.315	2.103	Parton Shower

Table 3: In Case 1, all environment metrics are OK but poor transfer results are observed, indicating a likely problem caused by NetBASILISK. Case 2 shows an example where the Source is the likely cause of the poor transfer result.

References

[1] Julian Jang-Jaccard and Surya Nepal. “A survey of emerging threats in cybersecurity”. In: *Journal of Computer and System Sciences* 80.5 (2014). Special Issue on Dependable and Secure Computing, pp. 973–993. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2014.02.005. URL: https://www.sciencedirect.com/science/article/pii/S0022000014000178.

[2] The Center for Strategic and International Studies. *Significant Cyber Incidents*. URL: https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents.

[3] ESnet-Team. *Energy Sciences Network*. Feb 2020. URL: http://www.es.net/.

[4] Eli Dart et al. “The Science DMZ: A Network Design Pattern for Data-Intensive Science”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: 10.1145/2503210.2503245. URL: https://doi.org/10.1145/2503210.2503245.

[5] Wikipedia Community. *DMZ (networking)*. 2021. URL: https://en.wikipedia.org/wiki/DMZ_(computing).

[6] Eric Boyd and Katarina Thomas. *NetBASILISK Project*. URL: https://netbasilisk.net/about.

[7] The ATLAS Collaboration. “The ATLAS Experiment at the CERN Large Hadron Collider”. In: *Journal of Instrumentation* 3.08 (2008), S08003–S08003. DOI: https://doi.org/10.1088/1748-0221/3/08/s08003.

[8] A A Ayllon et al. “FTS3: New Data Movement Service For WLCG”. In: *J. Phys.: Conf. Ser* 513.3 (2014), p. 032081. URL: http://stacks.iop.org/1742-6596/513/i=3/a=032081.

[9] F Donno et al. “Storage resource manager version 2.2: Design, Implementation, and Testing Experience”. In: *Journal of Physics: Conference Series* 119.6 (July 2008), p. 062028. DOI: 10.1088/1742-6596/119/6/062028.

[10] A Frohner et al. “Data Management in EGEE”. In: *Journal of Physics: Conference Series* 219.6 (Apr. 2010), p. 062012. DOI: doi:10.1088/1742-6596/219/6/062012.

[11] Globus Team. *Globus GridFTP*. URL: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/.

[12] A Dorigo et al. *XROOTD - A Highly Scalable Architecture For Data Access*. URL: https://xrootd.slac.stanford.edu/presentations/xpaper3_cut_journal.pdf.

[13] AGLT2 Personnel. *ATLAS Great Lakes Tier 2*. URL: https://www.aglt2.org/index.php.

[14] ESnet Team. *The My ES net portal*. 2021. URL: https://my.es.net/.

[15] Wikipedia Community. *Multi-Protocol Label Switching*. 2021. URL: https://en.wikipedia.org/wiki/Multiprotocol_Label_Switching.

[16] Mathias Kettner. *Check_Mk*. URL: https://mathias-kettner.de/check%5C_mk.html.

[17] Jem Guhit. *CA Performance Environment Metrics Script*. URL: https://github.com/NetBASILISK/EnvironmentMonitoring/blob/main/Scripts/RBIN_ind.py.

[18] Grafana Team. *Grafana is the Open Source Analytics and Monitoring Solution for Every Database*. URL: https://grafana.com.

[19] Jem Guhit. *Grafana Environment Metrics Script*. URL: https://github.com/NetBASILISK/EnvironmentMonitoring/blob/main/Scripts/AGLT2CHI_ind.py.

[20] P Fuhrmann. *dCache, The Overview*. URL: http://www.dcache.org/manuals/dcache-whitepaper-light.pdf.

[21] Jem Guhit. *Check MK Environment Metrics Script*. URL: https://github.com/NetBASILISK/EnvironmentMonitoring/blob/main/Scripts/AGLT2_ind.py.

[22] Humio Team. *Humio*. URL: https://www.humio.com.

[23] Jem Guhit. *Grafana Environment Metrics Script*. URL: https://github.com/NetBASILISK/EnvironmentMonitoring/blob/main/Scripts/AGLT2CHI_ind.py.

[24] Andreas Hanemann et al. "PerfSONAR: A Service Oriented Architecture for Multi-domain Network Monitoring". In: *Service-Oriented Computing - ICSOC 2005*. Ed. by Boualem Benatallah, Fabio Casati, and Paolo Traverso. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 241–254. ISBN: 978-3-540-32294-8.

[25] Ansible Team. *Red Hat Ansible*. URL: https://www.ansible.com.

[26] Elastic Team. *Elastic Stack*. URL: https://www.elastic.co/elastic-stack.

[27] Andrew Lake. *perfSONAR archiving-sandbox*. URL: https://github.com/perfsonar/archiving-sandbox.