trellis — An Analytics Framework for Understanding Slingshot Performance

Madhu Srinivasan, Dipanwita Mallick, Kristyn Maschhoff, Haripriya Ayyalasomayajula
AI and Advanced Productivity
Hewlett Packard Enterprise

madhu.srinivasan@hpe.com
dipanwita.mallick@hpe.com
kristyn.maschhoff@hpe.com
haripriya.ayyalasomayajula@hpe.com

Abstract—The next-generation HPE Cray EX and HPE Apollo supercomputers with Slingshot interconnect are breaking new ground in the collection and analysis of system performance data. The monitoring frameworks on these systems provide visibility into Slingshot's operational characteristics through advanced instrumentation and transparency into real-time network performance. There still exists, however, a wide gap between the volume of telemetry generated by Slingshot and a user's ability to assimilate and explore this data to derive critical, timely, and actionable insights about fabric health, application performance, and potential congestion scenarios. In this work, we present *trellis* — an analytical framework built on top of Slingshot monitoring APIs. The goal of *trellis* is to provide system-administrators and researchers insight into network performance, and its impact on complex workflows that include both AI and traditional simulation workloads. We also present a visualization interface, built on *trellis*, that allows users to interactively explore through various levels of the network topology over specified time windows, and gain key insights into job performance and communication patterns. We demonstrate these capabilities on an internal Shasta development system and visualize Slingshot's innovative congestion-control and adaptive-routing in action.

Index Terms—Performance of Systems, Machine learning, Network monitoring, Network topology, Information Visualization, Visualization systems and software, Visualization techniques and methodology

1 Introduction

The overall performance of applications is highly dependent on many factors such as system resources, health and availability of network interconnect, etc. Contention of shared resources, faulty components, imbalanced workload decompositions, can all significantly degrade performance. System administrators and application developers often encounter slow running jobs and network congestion ends up being one of the common root causes in such scenarios. The HPE/Cray Slingshot network employs various hardware mechanisms which are used to significantly reduce congestion within the network and to address network performance issues when problems do arise. System administrators and application developers can benefit from insights derived from a combination of job-related information from the workload manager and network telemetry data to debug these issues.

The HPE Cray EX supercomputer currently provides two tightly integrated monitoring frameworks, the Cray System Manager (CSM) [1] and the HPE Performance Cluster Manager (HPCM) [2] which consolidate telemetry data from different subsystems such as the network fabric, job management, storage, power, user applications and compute, and persist them in dedicated data stores.

However, additional analytical tools are needed beyond what the monitoring framework can provide, in order to investigate the sensitivity of application performance to network characteristics. Researchers and system administrators have to spend a significant amount of time cleaning, refining, and transforming the raw telemetry data to extract actionable insights. There are many efforts and tools available to address the data collection process. However, there are but a few end-to-end frameworks that target HPC systems, such as the ones in [3]–[6], that can specifically support end-users through data collection, preparation, fusion, and aggregation, all the way through to building robust data visualization and analytical pipelines. Such tools will shorten the time-to-insight for understanding various aspects of the high-speed network fabric. This ability to detect and adapt to network topology and traffic patterns is a valuable capability to facilitate application performance, as highlighted by the Exascale Computing Project [7].

Our solution, *trellis*, is designed to address the gap between data collection and getting useful insights from them. *trellis* enables users to easily extract data from the system, load the data into a highly optimized columnar data store for faster querying, perform a series of preprocessing on the data and finally delivering interactive and intuitive visualizations for fast real-time system monitoring and analysis.

In this paper, we discuss the architecture and features of *trellis* in detail. We also describe several examples of how

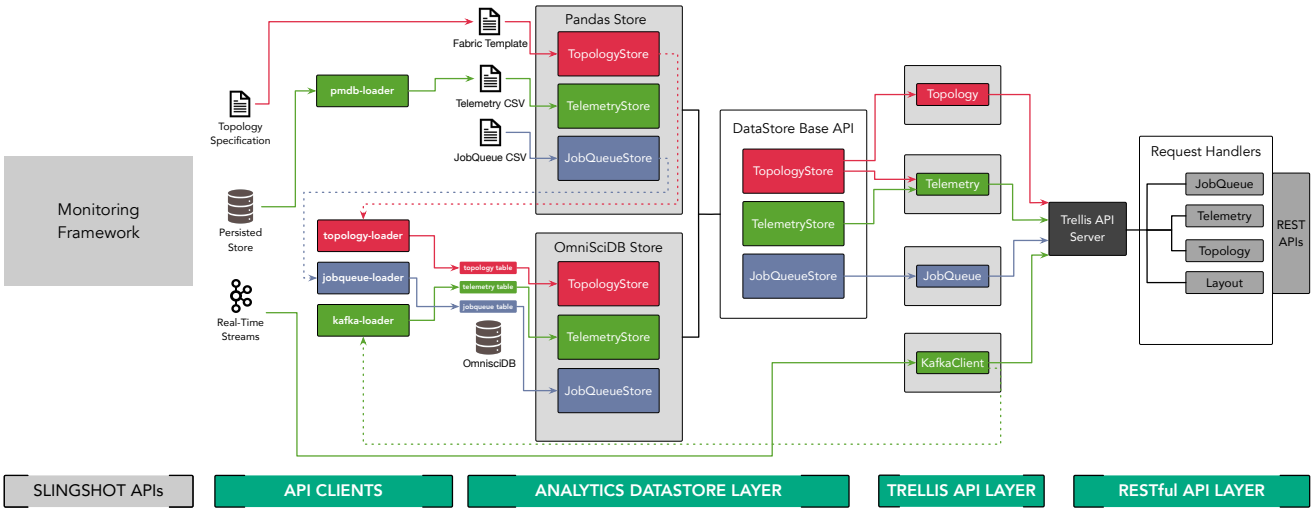


Fig. 1: Architecture overview of *trellis*. *trellis* optimizes access to Slingshot telemetry through its APIs, so users can easily gain insight into Slingshot's performance.

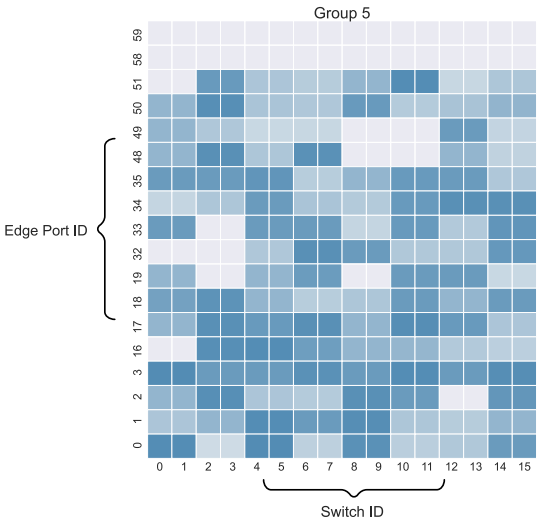


Fig. 2: Overview of a heatmap for Group 5. The entries of the heatmap can be color-coded to show telemetry or application specific metrics.

we use *trellis* to build visualizations to correlate network performance and application characteristics. One of our design goals for *trellis* is to facilitate ease of use within a data science or machine learning development environment, such as Jupyter Notebooks.

2 Architecture Overview

Figure 1 shows a high-level overview of the architecture of *trellis*. The framework includes four distinct functional components:

a) *API Client Layer*
b) *Analytics Datastore Layer*
c) *Trellis API Layer*
d) *RESTful API Layer*

The *API Client Layer* provides an adapter to acquire data from Slingshot telemetry APIs on the monitoring framework. The *Analytics Datastore Layer* is tasked with combining and persisting fabric performance data into an analytics-friendly format. The *Trellis API Layer* and *RESTful API Layer* implement and expose compute-intensive analytics functionalities via user-friendly Pythonic and HTTP REST APIs.

3 Monitoring the Network

Jobs not performing as expected is an issue that application developers and system administrators often have to diagnose. A slow network is often one of the root causes in such cases. Therefore, it is imperative to have a clear understanding of what is happening in the network at a given time. Monitoring and comparing the status of the network and traffic generated by job mixes are some of the most constructive ways to determine the root cause of a slow network.

trellis enables users to map bandwidth and congestion to different traffic patterns generated by jobs in the network, thereby helping them to unveil relationships between different job-types and placements, and network performance. In the following sections, we present our custom visualizations built on top of *trellis* to demonstrate how we can overlay job-related information on top of the network data to identify a few key communication patterns, and at the same time understand the state of the network with respect to congestion and bandwidth utilization.

3.1 Setup

To demonstrate the effectiveness of *trellis* in monitoring network performance alongside application performance, we instrument GPCNeT [8] and LAMMPS [9] using *CrayPat* [10] and execute them on a 1024 node Cray EX developmental system called *Shandy*. GPCNeT is a well known network benchmark application that uses MPI to generate a variety of congestor traffic patterns in the network (e.g. pairwise all-to-all and incast). Network telemetry is collected using Shandy's CSM framework and *trellis*, whereas application specific data such as node allocations and average MPI latencies are collected through *CrayPat*. The visualizations

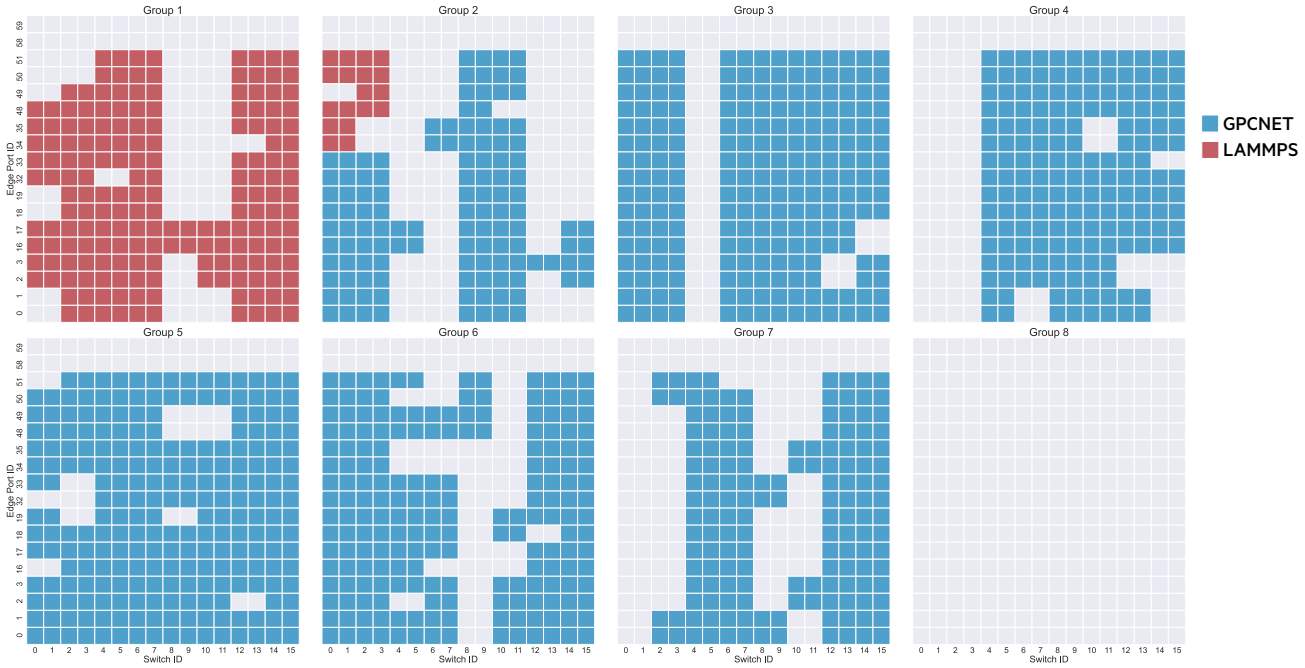


Fig. 3: A network map showing applications (color coded) executing at the edge ports. The edge ports map directly to compute nodes allocated to the applications.

are built on 4-hour telemetry data captured at 1 Hz sampling rate and persisted in OmniSciDB database for *post-hoc* analysis.

When we conducted our experiments, Shandy had Shasta v1.3 software installed. Each dual-socket compute node on Shandy has two 64-core EPYC Rome processors with a total of 256 GB DDR-4 memory per node. Its network topology consist of 8 groups, with 16 switches per group, and 128 nodes connected at each group. Each compute node in the system hosts dual Mellanox ConnectX-5 Network Interface Cards.

For the application runs, we use "out-of-the-box" build settings for both GPCNeT and LAMMPS. A total of 640 nodes were available at the time of the experiment, with GPCNeT running across 540 nodes, with 128 processes-per-node, and LAMMPS (2D Lennard-Jones diffusion coefficient calculation) running across 100 nodes, with 4 processes-per-node with OpenMP enabled at each process rank. Disk I/O was negligible for both applications.

3.2 About the Visualizations

To map telemetry and application performance across network groups, switches and ports, we use a heatmap as the key graphical unit for our visualizations. Figure 2 shows an example heatmap of a switch group 5 with rows representing the Port IDs and columns representing the Switch IDs. Each block in the heatmap can be color coded to show

- if an application was allocated to run at that port (for edge ports)
- an aggregated application performance metric (i.e. MPI latencies)
- an aggregated network telemetry statistic (i.e. bandwidth and congestion)

All values are aggregated over the time period of the application run. The key network statistics measured at each port are received bandwidth - `rxBW`, transmitted bandwidth - `txBW` and average frames blocked per second – `rxBlocked`. Additionally, for every edge port, we can show its associated compute node id.

3.3 Monitoring the network at the application level

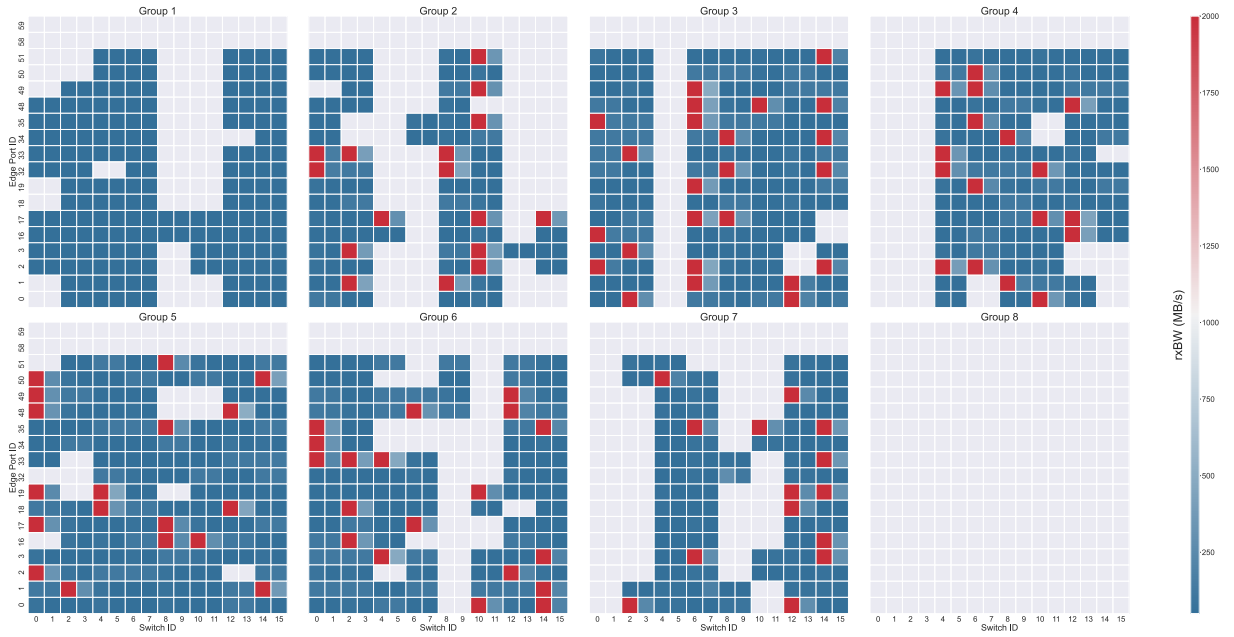
Figure 3 shows the node allocations for both LAMMPS and GPCNeT across all the groups on *Shandy*. Figures 4a and 4b show average injected bandwidth and average blocked frames, respectively, at the edge ports where both applications were executing. Further, the heatmaps indicate that congestion and bandwidth utilization across edge ports running GPCNeT are much higher compared to the edge ports assigned to LAMMPS.

In the next two sections, we focus on identifying communication patterns generated by GPCNeT.

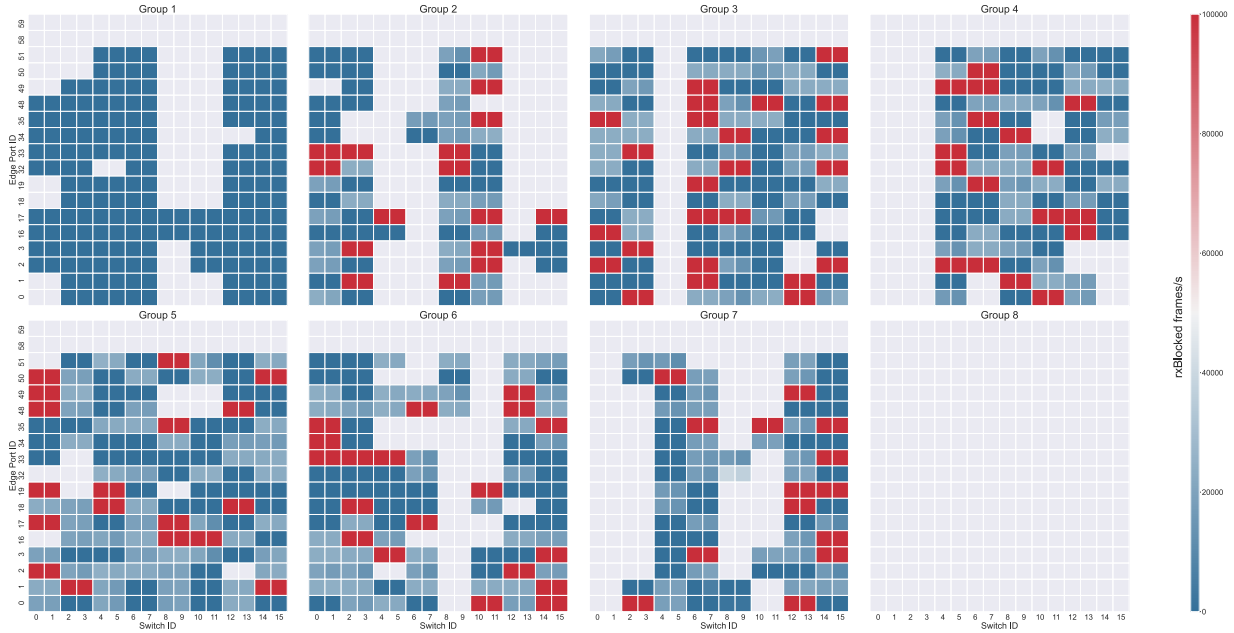
3.3.1 Identifying an Incast pattern

The heatmap in Figure 3 shows the edge ports (shaded blue) where GPCNeT is executing across groups 2-7. As part of its benchmark suite, GPCNeT induces both end-point congestion and intermediate congestion in the network.

Figure 5a shows the average MPI latencies across all ranks of GPCNeT. A closer inspection of the heatmap reveals a high MPI latency at a pair of ports from switches 8 and 9 in group 7. These two ports are connected to a single node through two network interfaces. Correspondingly, Figure 5b shows the average transmitted bandwidth for all the edge ports where GPCNeT was running. We notice a similar outlier at the same corresponding pair of ports i.e. port 0 of switches 8 and 9, in group 7, indicating higher-than-normal transmitted bandwidth. Now, for any



(a) Bandwidth injected at the edge ports by GPCNeT and LAMMPS, averaged across the application execution time.



(b) Average congestion, measured as *blocked frames per second* while receiving at edge ports. Edge ports used by GPCNeT (figure 3) appear congested.

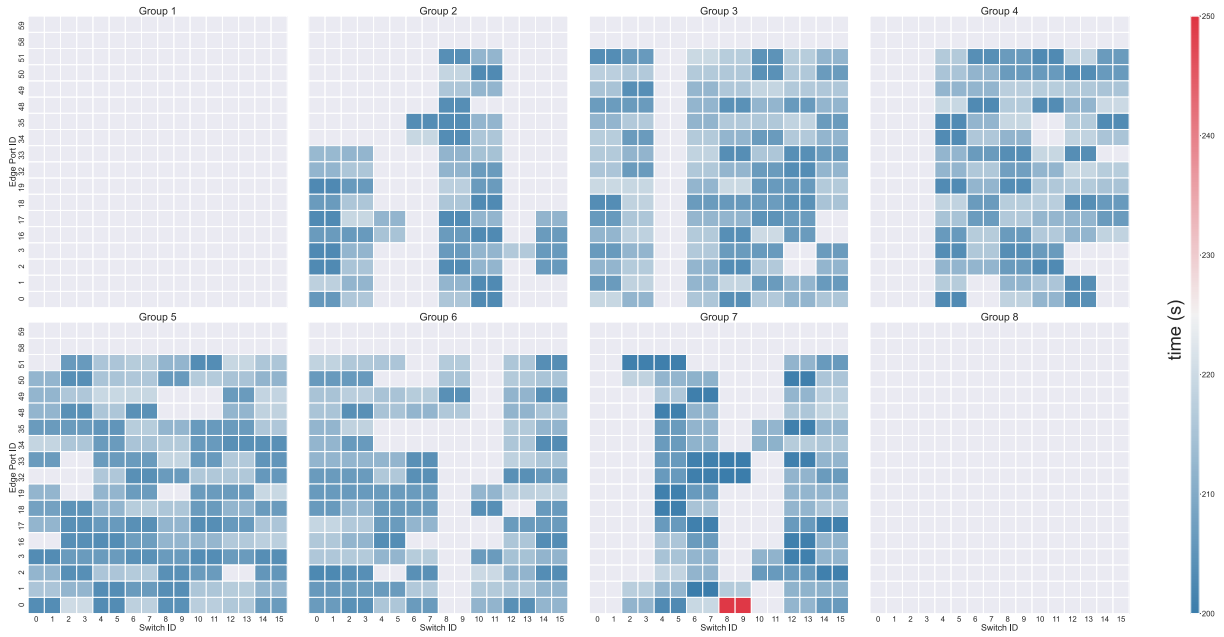
Fig. 4: Visualizing received bandwidth(top) and congestion(bottom) at the edge ports. *trellis* makes it easier to summarize and map large volumes of telemetry data into visuals such as these.

given edge port, transmitted bandwidth corresponds to bandwidth received at the node connected to that edge port.

Our hypothesis from this observation is that the higher MPI latencies correspond to an incast pattern observed in the same network location (i.e. switch, and port). We can qualitatively confirm this hypothesis as we know that GPCNeT indeed generates an incast congestion as part of its benchmark suite.

3.3.2 *Adaptive routing in action*

The heatmap in Figure 6a shows the transmitted bandwidth for global ports when GPCNeT was running. High bandwidth values are color-coded as red. In contrast, Figure 6b shows the transmitted bandwidth for the global ports when GPCNeT was not running. Immediately, one can observe a non-trivial bandwidth in the global links of group 8, in Figure 6a. If we recall, GPCNeT was executing only across groups 2 through 7. The appearance of transmitted bandwidth values in group 8, despite the fact that no jobs



(a) A map of average MPI latencies as measured at edge ports associated with GPCNeT.

(b) Average bandwidth egress as measured at edge ports associated with GPCNeT.

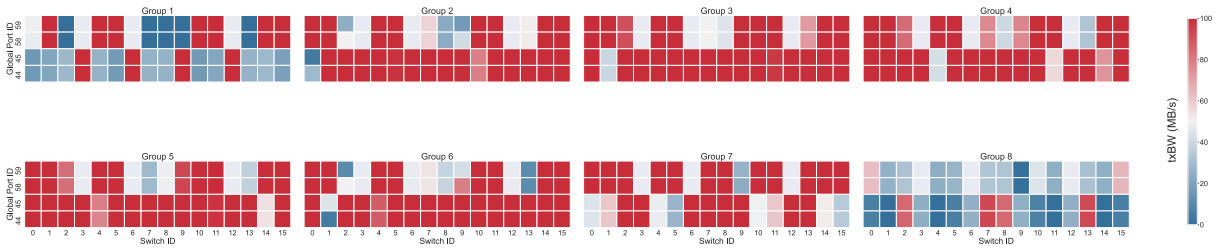
Fig. 5: Visualizing MPI latencies (top) and `txBW` (bottom) at the edge ports for GPCNeT. The high MPI latency observed in Switches 8 and 9 in Group 7 correspond to a high value of received bandwidth at the same switches in Group 7. This is indicative of incast congestion pattern generated by GPCNeT.

were allocated in that group, is indicative of non-minimal-path adaptive-routing in action.

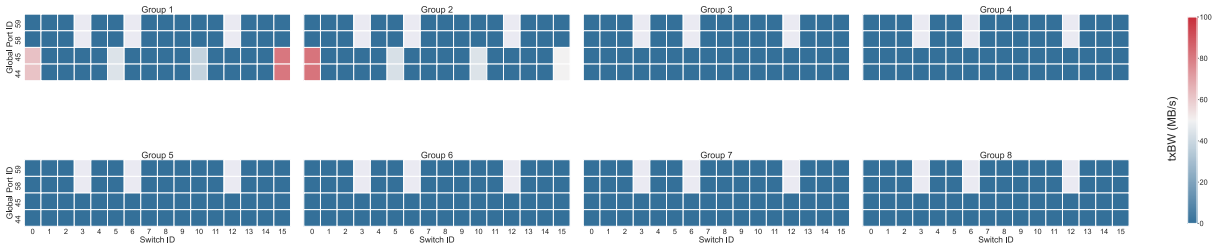
HPE's Slingshot interconnect employs adaptive routing features to avoid congestion impacts by routing traffic around hot spots in the network [11]. In figure 6a, GPCNeT causes congestion in the global links of groups 2-7, and therefore, Slingshot re-routes some of the global traffic through group 8.

3.4 Monitoring the network fabric

In addition to facilitating users to monitor the network at the application level, *trellis* can also be used to monitor the status of the overall fabric at any given point in time. We overlay network telemetry (bandwidth and congestion) on top of the network topology, and present a timeline of which jobs were executing in the system, where applicable. We developed a prototype web-based visualization interface for this purpose. Our key contribution with this design is the ease with which users can filter and apply thresholds across



(a) A map of average bandwidth egress at global ports across all groups, during a GPCNeT run.



(b) A map of average bandwidth egress at global ports across all groups, when GPCNeT is not running.

Fig. 6: Visualizing bandwidth egress at global links. There is a non-trivial amount of traffic being routed through global links in Group 8 even when GPCNeT is only running in Groups 2-7 (top). We do not see this phenomena when GPCNeT is not running (bottom).

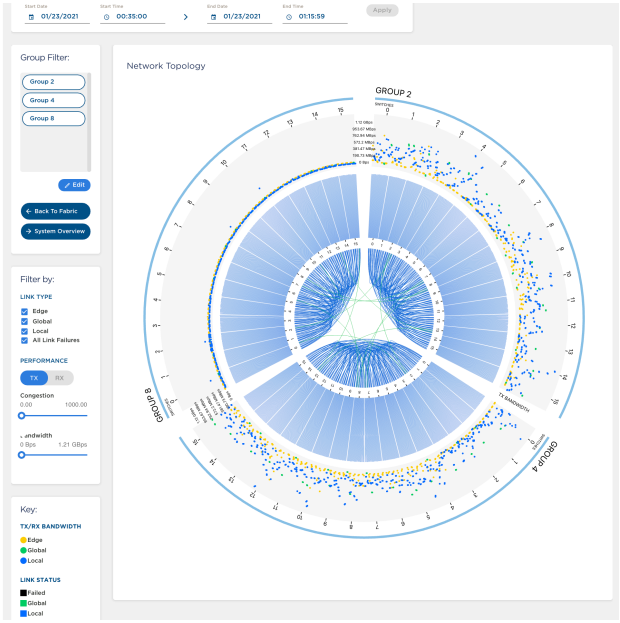


Fig. 7: A web user interface showing a chord visualization of the network topology, along with aggregated metrics across all ports. The metrics are color coded by port type.

both time and metrics, at different hierarchies of the network topology, and obtain highly intuitive and interactive visualizations to enable useful insights into the status of the network.

Figure 7 shows an example of the interface presenting a topological representation of the network along with the network traffic visualized in the form of a chord diagram [12]. The web interface presents several filters and options. A user can begin by selecting a time range using the date-time filter on the top and a metric of interest from the items on the left. Additionally, the user can select various network entities such as group, link types (edge, local, and global) and port types (edge, local and, global). Upon submitting the selections, the chord diagram showing the topological view gets updated with annotated data.

The sectors in the chord diagram, in figure 7, correspond to a selected subset of groups from the system. Each group is again subdivided into smaller sectors representing different switches. An aggregated metric value is shown in the outer-most segment across all the ports in a group. The values are color coded by port types: yellow for edge ports, blue for local ports and green for global ports. The inner-most segment of the chord diagram depicts the inter and intra group connectivity. Blue arcs indicate local links and the green arcs correspond to global links. For the visualization shown in the figure, metric values for each network entity have been averaged over the selected time period. However, additional aggregation methods such as `max`, `min`, and `sum` are also available.

Figures 8a, 8b and 8c show progressive views available from the interface. A user can interact with the network data at different levels of granularity, starting from a high-level group view, through a switch-based view, down to viewing metrics at a single port. Any update on the selection panel modifies all the views of the interface.

In order to tackle the complexities of a hierarchical network topology, *trellis* performs aggregations on the metrics at different levels of the hierarchy (group, switch and port). These aggregated views are made available to the users through interactive visualizations.

4 Implementation

The core libraries in *trellis* have been implemented in Python. We used Plotly [13] and matplotlib [14] to create the

(a) System Overview **(b)** Switch View **(c)** Port View

Fig. 8: Our web-based user interface with three different views that progressively visualizes three different levels of the network hierarchy. Figure 8a shows the transmitted bandwidth across all the groups. Figure 8b shows the transmitted bandwidth for all the ports within a single switch, along with job information. The visualization in figure 8c allows the user to observe the trend of transmitted bandwidth over time for a particular port at a switch.

heatmaps on top of *trellis*, to monitor and correlate network utilization with applications (Section 3.3). Dash [15], a web interface library, was used to display these visualizations. To build the visualization interface for monitoring the overall status of the network (Section 3.4) we used React [16] and d3.js [17].

trellis can also be used from within a Jupyter Notebook environment. Figure 9 illustrates an example of using the *trellis* API to obtain the connectivity information for all global ports as an adjacency list. The output is a pandas dataframe, which can be used for further analysis and investigation. We believe that having the flexibility to use *trellis* within the datascience ecosystem of tools will lower the barrier-of-entry and enable researchers to use the framework as a building block for more complex analytics and machine learning pipelines.

5 Conclusion

This paper introduces *trellis*, an analytics framework to observe and understand network performance. As core pieces in the design of *trellis*, we present two key components that address the challenges of aggregating and analyzing high-throughput and high-volume network telemetry and making them easily available to end-users.

- An industry-standard columnar database, OmniSciDB [18], to efficiently store and quickly analyze large streams of telemetry data.
- A web-standard compliant RESTful API server, based on TornadoWeb [19], that exposes rich and complex analytical queries on telemetry streams to enable web-based interactive visualization front-ends.

The architecture of *trellis* is designed as a component-based system that allows users to easily integrate it in their existing data science and visualization workflows.

We have also presented how users can investigate network communication patterns generated by their applications (eg. GPCNeT, LAMMPS), and correlate network performance with application performance characteristics on a Slingshot-based system. In addition, we also demonstrated a web-based user interface that successfully fuses network telemetry information with job statistics and allows users to evaluate the health of the network at different hierarchical levels (i.e. group, switch and port).

6 Future Work

At the time of writing this paper, we have successfully deployed *trellis* on a 1024 node system (*Shandy*). However, scaling up *trellis* to work on larger systems, and eventually on exascale systems will pose unique challenges when trying to gain insights from large volumes of telemetry data. For large scale systems, raw telemetry data is likely to grow to several petabytes, even with a 30-day retention period, for instance.

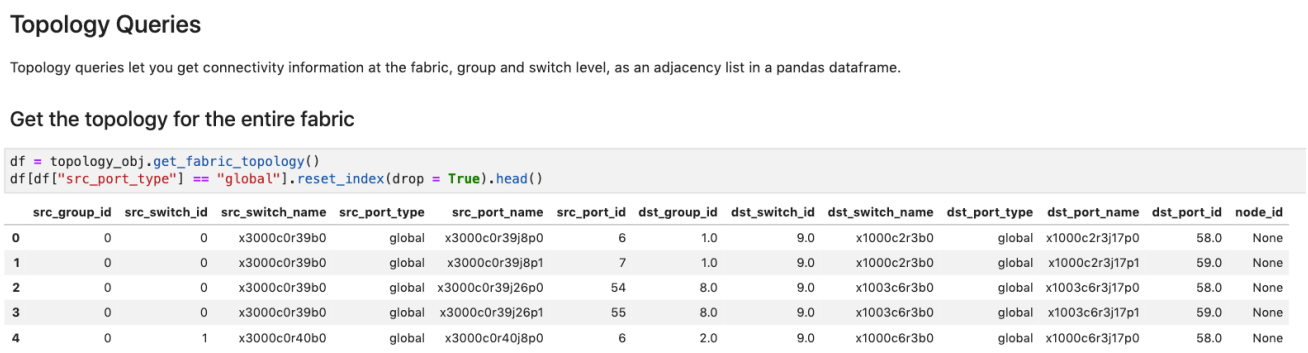
Currently, we employ techniques such as aggregating data at various levels of granularity, filtering and thresholding performance metrics, and resampling metrics at coarser temporal granularity. But there exists a trade-off between interactivity and computation time which becomes prominent as the scale of the data increases. Therefore, in order to build visualizations for large and high-resolution data, we need advanced data management, analysis and visualization techniques. We are working on enhancing our tools and interfaces so that end-users can continue exploring and interacting with the data to gain timely insights in an efficient manner.

For future work, we plan to investigate advanced machine learning models based on *trellis*. These models, once successful, can associate and predict job execution times with network performance.

References

[1] J. M. Brandt, C. J. Brown, S. G. Donoho, A. C. Gentile, J. Greenseid, W. Kramer, P. Langer, A. Rashid, K. Rhem, and M. Showerman, "Exploring New Monitoring and Analysis Capabilities on Cray's Software Preview System (Final Version)." *Proceedings of CUG 2019: Cray User Group*, 5 2019.

[2] HPE, "HPE Performance Cluster Manager," https://h20195.www2.hpe.com/v2/gethtml.aspx?docname=a00044858enw, 2021, Accessed: 17-03-2021.



Topology Queries

Topology queries let you get connectivity information at the fabric, group and switch level, as an adjacency list in a pandas dataframe.

Get the topology for the entire fabric

```
df = topology_obj.get_fabric_topology()
df[df["src_port_type"] == "global"].reset_index(drop = True).head()
```

	src_group_id	src_switch_id	src_switch_name	src_port_type	src_port_name	src_port_id	dst_group_id	dst_switch_id	dst_switch_name	dst_port_type	dst_port_name	dst_port_id	node_id
0	0	0	x3000c0r39b0	global	x3000c0r39j8p0	6	1.0	9.0	x1000c2r3b0	global	x1000c2r3j17p0	58.0	None
1	0	0	x3000c0r39b0	global	x3000c0r39j8p1	7	1.0	9.0	x1000c2r3b0	global	x1000c2r3j17p1	59.0	None
2	0	0	x3000c0r39b0	global	x3000c0r39j26p0	54	8.0	9.0	x1003c6r3b0	global	x1003c6r3j17p0	58.0	None
3	0	0	x3000c0r39b0	global	x3000c0r39j26p1	55	8.0	9.0	x1003c6r3b0	global	x1003c6r3j17p1	59.0	None
4	0	1	x3000c0r40b0	global	x3000c0r40j8p0	6	2.0	9.0	x1000c6r3b0	global	x1000c6r3j17p0	58.0	None

Fig. 9: Using *trellis* from within a jupyter notebook.

[3] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 154–165.

[4] T. Fujiwara, J. K. Li, M. Mubarak, C. Ross, C. D. Carothers, R. B. Ross, and K.-L. Ma, "A visual analytics system for optimizing the performance of large-scale networks in supercomputing systems." *Visual Informatics*, vol. 2, no. 1, pp. 98–110, 2018.

[5] J. K. Li, M. Mubarak, R. B. Ross, C. D. Carothers, and K. L. Ma, "Visual Analytics Techniques for Exploring the Design Space of Large-Scale High-Radix Networks," *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, vol. 2017-Septe, pp. 193–203, 2017.

[6] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P. T. Bremer, "Analyzing Network Health and Congestion in Dragonfly-Based Supercomputers," *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pp. 93–102, 2016.

[7] M. A. Heroux, J. Carter, R. Thakur, J. S. Vetter, L. C. McInnes, J. Ahrens, T. Munson, and J. R. Neely, "ECP Software Technology Capability Assessment Report," 2 2020.

[8] S. Chunduri, T. Groves, P. Mendygral, B. Austin, J. Balma, K. Kandalla, K. Kumaran, G. Lockwood, S. Parker, S. Warren, N. Wichmann, and N. Wright, "Gpcnet: Designing a benchmark suite for inducing and measuring contention in hpc networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356215

[9] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002199918571039X

[10] L. DeRose, B. Homer, and D. Johnson, "Detecting application load imbalance on high end massively parallel systems," in *Euro-Par 2007 Parallel Processing*, A.-M. Kermarrec, L. Bougé, and T. Priol, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 150–159.

[11] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[12] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006.

[13] P. T. Inc. (2015) Collaborative data science. Montreal, QC. [Online]. Available: https://plot.ly

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[15] P. T. Inc. (2020) Collaborative data science. Montreal, QC. [Online]. Available: https://plotly.com/dash/

[16] [Online]. Available: https://reactjs.org/

[17] M. Bostock. (2012) D3.js - data-driven documents. [Online]. Available: http://d3js.org/

[18] OmniSciDB, "OmniSci, Open Source Analytical Database and SQL Engine," https://www.omnisci.com/platform/omniscidb, 2020, Accessed: 20-08-2020.

[19] TornadoWeb, "Tornado Web Server," https://www.tornadoweb.org/en/stable, 2020, Accessed: 20-08-2020.