Eurographics Conference on Visualization (EuroVis) 2018
J. Heer, H. Leitte, and T. Ropinski
(Guest Editors)

Volume 37 (2018), Number 3

Interactive Investigation of Traffic Congestion on Fat-Tree Networks Using TREESCOPE

H. Bhatia[1], N. Jain[1], A. Bhatele[1], Y. Livnat[2], J. Domke[3], V. Pascucci[2], and P.-T. Bremer[1,2]

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA
[2]Scientific Computing & Imaging Institute, The University of Utah, Salt Lake City, UT, USA
[3]Tokyo Institute of Technology, Tokyo, Japan

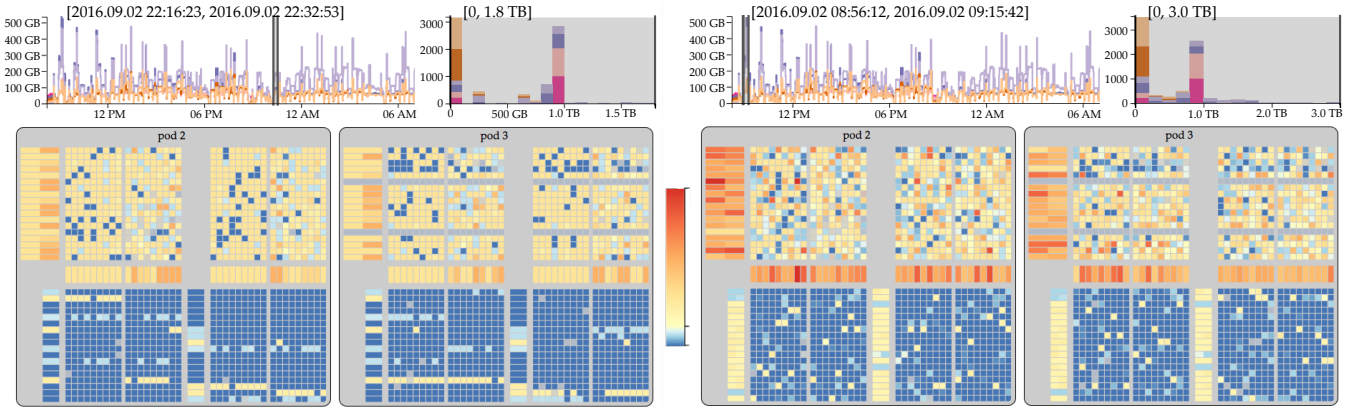

Figure 1: TREESCOPE *enables interactive and unified exploration of network traffic for large-scale fat-tree networks, including the visual analytics of network counters, job queue logs, job placements, and routing scheme. The figure shows the network traffic during the execution of the same application using two routing schemes,* free *routing (left) and* SAR *scheme (right). The visualization shows temporal and distributional statistics (top), and detailed per-link traffic on half of the 1296-node fat-tree cluster in use (bottom). The* free *routing distributes the traffic more uniformly (average traffic maps to yellow) and is about 15% faster than the* SAR *scheme.* TREESCOPE *helps users explore the data and formulate hypotheses on the causes for performance degradation, such as the presence of hotspots in the traffic on the right.*

Abstract

Parallel simulation codes often suffer from performance bottlenecks due to network congestion, leaving millions of dollars of investments underutilized. Given a network topology, it is critical to understand how different applications, job placements, routing schemes, etc., are affected by and contribute to network congestion, especially for large and complex networks. Understanding and optimizing communication on large-scale networks is an active area of research. Domain experts often use exploratory tools to develop both intuitive and formal metrics for network health and performance. This paper presents TREESCOPE*, an interactive, web-based visualization tool for exploring network traffic on large-scale fat-tree networks.* TREESCOPE *encodes the network topology using a tailored matrix-based representation and provides detailed visualization of all traffic in the network. We report on the design process of* TREESCOPE*, which has been received positively by network researchers as well as system administrators. Through case studies of real and simulated data, we demonstrate how* TREESCOPE*'s visual design and interactive support for complex queries on network traffic can provide experts with new insights into the occurrences and causes of congestion in the network.*

CCS Concepts

•***Human-centered computing*** → ***Visualization application domains; Visual analytics;***

1 Introduction

High-performance computing (HPC) is an integral component of the modern scientific workflow. In order to support the growing scale and complexity of scientific and engineering applications, HPC facilities are constantly optimizing the use of available resources. One of the primary performance bottlenecks in large-scale applications is the communication between distributed nodes. Hence, understanding the implications of the design and configuration of the large-scale network interconnect on current and future applications is crucial for optimally utilizing the existing facilities as well as for planning and procuring next-generation machines. A particularly important design parameter is the *network topology*, which defines how thousands of computational nodes are interconnected to form a supercomputer. Not only can the network topology heavily influence the performance of certain applications, but it also represents a hard constraint, as once installed, it usually cannot be changed. Therefore, all other systemwide design choices must be compatible with the network topology to obtain the best performance in HPC applications.

Interconnect-related performance degradation on a network topology can have many causes, and is often unique to a particular configuration, state of the network, and even input parameters of active applications. Communication-based slowdowns can occur due to various factors, such as ill-suited *routing schemes* or *job-placement policies* (mapping of applications to computational nodes), especially if either is incompatible with the underlying topology [JBW*16]. Therefore, developing automatic techniques to find and diagnose network problems has been challenging, and remains an active area of research.

The lack of automated tools and the exploratory nature of the problem make visual analytics solutions an attractive choice. To develop such a system, it is crucial to understand the diverse and sometimes conflicting needs of all interested users. For example, network researchers are interested in developing new routing schemes, new network configurations, whereas system administrators are typically concerned with ongoing network health, optimizing utilization, and diagnosing failures. In general, both user groups are interested in visualization tools to understand network behavior with some specific use-cases including exploration of network traffic to identify congested and underutilized portions of the network, and to determine potential causes of problems.

Here, we focus on one of the most common network topology, the *fat-tree* topology [Lei85]. In particular, we aim to allow an interactive visual exploration of network congestion and its potential causes. Several sophisticated visualization solutions have been proposed for other topologies, such as torus [LLB*12] and dragonfly [BJL*16, LMR*17]. However, due to unique characteristics such as the strictly hierarchical nature of the topology, the fat-tree topology is significantly different from these topologies, and intuitive visualizations for fat-trees are still lacking. Instead, application specialists and system administrators typically diagnose unexpected behavior via manual exploration, mostly through aggregated statistics and/or simple, static visualizations, which not only is slow and fails to scale, but more importantly is error prone and disallows making detailed investigation.

Domain experts often have an intuitive expectation of how a well-behaved network should look, e.g., ideally the network should be evenly loaded, and *hotspots* (overloaded network links) may indicate bottlenecks. Key metrics to characterize the state of a network are the various hardware counters collected from the computational nodes or network switches. In particular, the number of packets sent over all network links or processed by all network ports represents the total traffic on the network. Although nonuniformly distributed packet counts may indicate bottlenecks or underutilization, analysis of its root causes typically requires supplementary information, e.g., job logs recording which applications ran on which nodes and/or routing schemes indicating potential sources or destinations of high packet volumes.

Contributions. We present TREESCOPE, a unified solution to investigate network traffic on supercomputers with fat-tree topology. Shown in Figure 1, TREESCOPE is an interactive web-based visualization tool that enables users to explore network traffic (and other relevant hardware counters) and investigate the effects of job-placement and routing schemes. TREESCOPE uses a tailored matrix-based graph encoding of the fat-tree topology, which provides a high-level overview while also supporting detailed queries for specific information. We discuss our collaborative design process, as well as the data and task abstractions, which will be relevant to other visualization researchers working in similar domains. Through two case studies using real and simulated data for applications of interest on leadership-class computing machines, we report a success story of how visualization research can be leveraged to support crucial inquiries in other fields.

2 Fat-Tree Networks

The fat-tree topology [Lei85] was designed to connect processors in parallel clusters and supercomputers. As commodity hardware became available to build systems with the fat-tree topology, they became popular and are widely used today to build medium- to large-scale supercomputers and infrastructure for data centers. Currently, about 50% of machines listed in the TOP500 list [MSDS] as well as several data centers [XZWX17] use fat-tree topology. The fat-tree network derives its name from its resemblance to a k-ary tree whose communication bandwidth increases as we get closer to the root. However, in practice, hardware cables provide a fixed amount of bandwidth, and network switches have a fixed number of ports. Thus, multiple switches are grouped together closer to the root to provide the increase in bandwidth. Typical implementations of the fat-tree topology are based on Clos networks [Clo53], and both terms are often used interchangeably.

Referring to Figure 2, we describe the construction of a three-level (the most common configuration) fat-tree using network switches with a fixed port count, k. The level 1 (L1), called the *edge layer*, is the lowest level of the tree. Half the ports of each L1 switch are connected to $k/2$ compute nodes. The remaining ports are connected to $k/2$ switches in level 2 (L2), also known as the *aggregation layer*. The L1 switches connected to L2 switches form a fully connected bipartite graph, commonly called a *pod*. Pods are connected in an all-to-all manner at level 3 (L3), also called the *core*

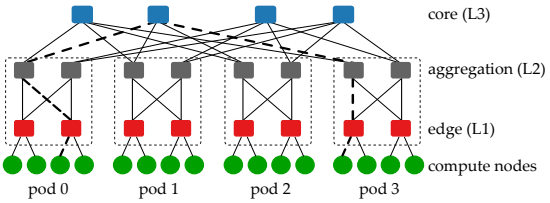

Figure 2: *A small fat-tree network for $k = 4$ contains 8 L1, 8 L2, 4 L3 switches, and up to 16 compute nodes. The L1 and L2 switches are logically grouped into four separate closely connected pods, and the L3 switches (from 2 bundles) connect different pods. The bold dashed line shows an example of the path between two compute nodes in different pods, which takes five hops.*

layer, through specialized hardware called *director-class switches* (referred hereon as *"bundles"* to avoid confusion with standard switches), such that L3 switches corresponding to each bundle connect to some of the L2 switches in each pod. A full-bisection, three-level fat-tree with equal bandwidth across levels, as shown in the figure, supports k pods, and is comprised of $k^2/2$ L1, $k^2/2$ L2, and $k^2/4$ L3 switches, and $k^3/4$ compute nodes. The pairs of nodes/switches are connected via a *bidirectional* link, which in practice is implemented using two independent physical cables.

3 Related Work

Several approaches exist for visualizing network traffic [IGJ*14]. One set of techniques focuses on the logical organization of the data, e.g., visualizing the flow of data between MPI ranks [HE91,HCR01,SMM*13]. Other approaches aim to visualize the impact of traffic on the hardware interconnect [LLB*12, BJL*16]. Here, we focus on the latter, since the goal is to identify hotspots/congestion in the physical network infrastructure, and investigate how different applications, routing schemes, and/or job placement policies may contribute to it. Designing visualizations for complex and multi-dimensional connectivity between network components is challenging, especially because visualization layouts are not transferable among different topologies due to their vastly different configurations. Several visualizations have been developed for n-dimensional tori [ABC*05], which can be naturally represented as regular meshes [LLB*12, MIB*14, TSW14, CDJM14]. Similarly, dragonfly topologies have been visualized with graph-based layouts [BJL*16, LMR*17].

Graph visualization. The visualization community has studied graph visualization in great detail [Ber67, PCJ97, BETT98, HMM00]. It is well accepted that *node-link visualizations*, which use lines to denote links between connected nodes, suffer significantly from scalability and occlusion [GFC04, GFC05]. Instead, *matrix-based representations* visualize the connectivity as adjacency matrices [Ber67]. Such representations have been shown to be better suited for almost all types of graph-specific queries [GFC04, GFC05], except route-finding, where the node-link representations can perform better for relatively small graphs. Matrix-based visualizations have been successfully in a wide variety of applications, e.g., visualization of social networks [HF06], telecommunication networks [BETT98], HPC networks [WCC*17], and brain networks [ABHR*13]. Considering the dense connectivity of the fat-tree topology, TREESCOPE uses a matrix-based layout to visualize the network.

Fat-tree visualization tools. Fat-tree specific tools include the Paraver visualization tool [Bar] as well as the Boxfish fat-tree module [BDM15]. Both use layered graph approaches with the corresponding scalability and occlusion issues. To alleviate some of the occlusion, Brown et al. [BDM15] propose to omit unused links. However, this can be misleading since unused links are not necessarily idle, but may be dysfunctional, which, especially for system administrators, makes them one of the more important aspects of a network. Zhou et al. [ZSC03] visualize the network connectivity as a symmetric adjacency matrix, with the nodes and switches grouped together to form both the rows and columns of the matrix. Two-way traffic can naturally be visualized on the matrix, while the transactions are depicted through 3D glyphs animation. However, such a representation suffers from sparsity and redundancy, since each switch and node appears twice, once as an input and once as an output. In practice, the sparsity is of particular concern as screen space is limited, and therefore large networks cannot be easily displayed. Furthermore, a pure matrix layout does not reflect the structure of the network, and it is difficult to determine the (logical) distance between two nodes or to understand the hierarchical organization of the switches. One particular requirement for TREESCOPE was to design a compact, yet complete, visual representation of the network topology.

4 Design Methodology

Designing an interactive visualization tool to support domain-specific inquiries is challenging, especially due to the gaps between the understanding and expectations of visualization scientists and domain experts [Wij06]. Fortunately, there exists extensive literature on methodologies for visualization design [AS05, Mun09, SMM12, BNTM16], guiding the process for a successful design by translating the domain knowledge and vocabulary into visualization terminology, and exploring various visualization choices suited to the application at hand.

We followed the four-phase nested model proposed by Munzner [Mun09], as it offers a clear distinction between different design phases as well as specific guidelines for appropriate validation. Due to space restrictions, this paper presents only the first three phases and omits the last phase (algorithm design).

TREESCOPE was developed through a collaborative process with domain experts, who were actively involved in our design process and provided continuous feedback on the evolving visualization design of TREESCOPE. These domain experts include three staff members at LLNL's HPC facility looking to monitor the network traffic on supercomputers, as well as four research scientists trying to understand the impact of the network on the communication performance of application codes (and vice versa). These experts also evaluated TREESCOPE through case studies, two of which are discussed in this paper.

4.1 Domain Problem Characterization

This section summarizes the first phase of our design process [Mun09]. Over a period of 6 months, we conducted

bi-weekly in-depth discussions with domain experts to learn about application domain, understand the issues and challenges they face, and gather a list of requirements. Generally, domain experts are interested in potential communication bottlenecks, and in how to improve the network throughput, and thus, the performance. Communication bottlenecks are often caused by *network congestion* or *hotspots*, which slow down communication routed through them, and in turn negatively impact application performance. Hotspots can be broadly defined as switches or links that have much higher traffic passing through them than the global average. Existing tools to identify congestion are limiting in both scale and functionality. Currently, experts typically explore aggregated statistics which can obscure many localized effects and rarely provide specific insights. Instead, descriptive visualizations that support an interactive exploration of the data are needed.

During the discussions, we jointly developed a list of specific types of data ($D\#$) the target design should support as well as a set of specific tasks ($T\#$) necessary to understand network congestion.

Exploration of network traffic. The primary use-case for TREESCOPE is the exploration of network traffic to identify congestion. To facilitate such exploration, the experts collect

($D1$) detailed connectivity information of the network, i.e, for each switch in the network, a list of all switches it was connected to through all active ports; and
($D2$) network counters for each port (captured every 30–90 seconds), such as the amount of data sent and received, the number of dropped packets, the aggregated wait time before packets were forwarded, as well as status flags and errors, such as buffer overload, etc.

Given the design of a fat-tree topology, detailed information for each switch, such as level and pod id can be derived from the connectivity information. For any of the network counters, one is typically interested in

($T1$) temporal statistics of traffic to determine if and when the network was underperforming;
($T2$) distributional statistics of traffic to find underutilized and/or congested network links;
($T3$) combined/filtered statistics and visualization to isolate the congestion with respect to level (e.g., the $L1 \rightarrow L2$ links) and/or directionality (e.g., the $L1 \rightarrow L2 \rightarrow L3$ links); and
($T4$) a simple and intuitive, yet complete visualization of traffic on all links and switches in the network, to facilitate visual examination of congestion.

Exploration of job execution. Visual identification of hotspots and/or underutilized links is only the first step toward the overall goal. If a bottleneck is observed, users are interested in identifying potential causes, e.g., which *jobs* (applications running on HPC machines) were creating these bottlenecks. Therefore, users collect

($D3$) a complete job history, i.e., a record of which jobs were running on which nodes along with their start and end times.

Using this auxiliary data, users want to complement the understanding of network traffic, and examine if any specific jobs were responsible for congestion. In particular, the tasks are to

($T5$) provide a simultaneous visualization of job placement on corresponding nodes on the network; and
($T6$) visualize traffic during the execution of selected job(s).

Exploration of traffic routing. For a root-cause analysis of where the congestion originates, it is important to explore the flow of traffic on the network. To this end, users collect

($D4$) routing tables, i.e., which ports were used to send packets between two nodes. We note that routing tables are dynamic, and can change for various reasons, such as switch, node, or link failures, experimental setups, etc.

Using time-varying routing tables, specific user queries include

($T7$) footprint of a (set of) job(s), i.e., all routes (potentially) used for communication by chosen job(s), useful for hypothesizing about the role of corresponding job(s) in creating congestion;
($T8$) footprint of a (set of) network component(s), i.e., the subset of the network reachable from a selected switch/port, useful for hypothesizing about the role of an overloaded port in creating congestion; and
($T9$) arbitrary combinations of the above.

Exploration of inter-job interference. A particularly important goal is to identify whether certain communication-heavy jobs interfered with other jobs' communication by congesting the network. This can be achieved through a combination of job-related and routing-related tasks defined above, i.e, ($T6$), ($T7$), and ($T8$).

4.2 Data and Task Abstraction

The next phase of the design process [Mun09] entails translating the scope and requirement for TREESCOPE into visualization vocabulary using appropriate abstractions, followed by validation with the domain experts. We summarize several of our discussions with the domain experts spanning a period of about 3 months.

First of all, it is important to note that the so-called "fat-tree topology" is a misnomer, as the underlying connectivity implies a graph, not a tree (see Figure 2). The *compute nodes* and *switches* in the network form the *nodes* of the graph, and *links* in the network correspond to graph's *edges*. Since network switches are uniquely associated with levels *L1*, *L2*, and *L3*, we define a concise notation for all graph nodes by denoting the compute nodes as *L0* nodes. We note that although a fat-tree topology may contain more than three levels, to the best of our knowledge, such a full-bisection bandwidth configuration with four or more levels has not been deployed in practice; hence, we focus TREESCOPE on fat-tree networks containing three levels only.

Recall from Section 2 that network links are bidirectional, and behave independently; therefore, we consider them as pairs of graph edges with opposite directionality. Aggregating bidirectional traffic into a single link, as done in similar tools for other types of network topology [LLB*12, BJL*16], can be misleading. Indeed, considering the two directions separately is important to our users because following the traffic direction may indicate which nodes are possibly creating bottlenecks, e.g., in cases when one direction of the link is loaded while the opposite is used sparingly. To establish a precise notion for bidirectional links, we call a given link $\{s \rightarrow d\}$ an *"up"* link if the level of the source node, s, is

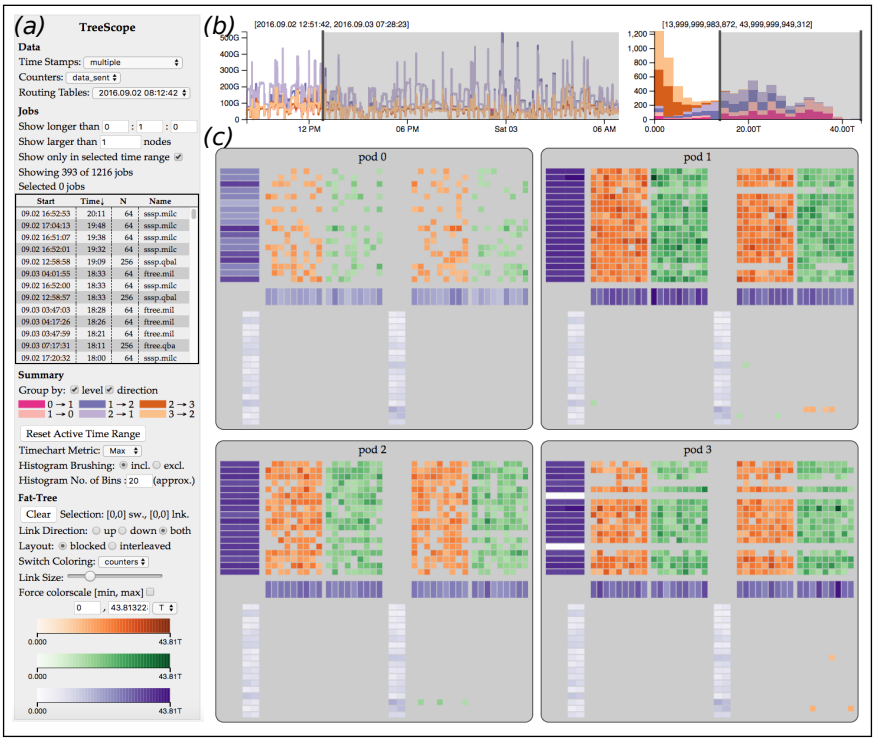

Figure 3: TREESCOPE *provides both overview and detailed visualizations of network traffic. The figure shows various components of* TREESCOPE. *(a) A complete UI panel is provided on the left with several options to enable interactive queries. (b) Summary overviews (temporal and distributional) are given at the top. (c) The central component is the detailed fat-tree view, showing per-link and per-switch traffic on adjacency matrices. The visible elements are filtered based on the active traffic range, which is selected using the brush in the histogram. Interactive selection and filtering is important to show heavily loaded links. Both summary plots are shown in six colors (grouped by level and direction), whereas three single-hue color maps are used for "up" links, "down" links, and switches, respectively.*

lower than that of the destination node, d. The reverse directions are correspondingly called *"down"* links.

Several different models for task abstraction have been proposed [AES05, LPP*06, HS12, BM13, BNTM16]. Here, we follow a multi-level typology [BM13] to characterize the scope and requirements of TREESCOPE. This approach allows translating domain-specific tasks into a sequence of interdependent abstract visualization tasks, using which their scope can be defined with respect to *why* and *how* a visualization task is performed, as well as *what* the task inputs and outputs are.

Why? The primary undertaking of TREESCOPE is to *discover* the presence and causes of network congestion, and its dependence upon factors like routing schemes and job placement. The required tasks aim to *explore* and *query* the data for trends and anomalies, as well as for comparison and summarization of data.

How? Next, we establish the encoding and manipulation of visualization tasks with respect to the corresponding data. ($T1$) and ($T2$) deal with time-series and distribution data; we choose 1D plots as they are well suited for the purpose and the domain experts are familiar with them. ($T3$) then becomes a simple *filtering* operation on ($T1$) and ($T2$). ($T4$) requires visualization of the entire network; our encoding for graph visualization is described in Section 5.1. Visualizing the job placement, i.e., ($T5$), is an *attribute-based task* [LPP*06], and can be performed by highlighting relevant nodes of the graph (discussed in Section 5.2). ($T6$) is also a filtering operation based upon the time range when a given job was active. The tasks related to routing are *browsing tasks* [LPP*06], and can be generally described as following certain paths using the active routing tables based on a known set of source/destination nodes ($T7$), a selected link ($T8$), or both ($T9$).

What? The lowest level descriptors of tasks further categorize them based on data semantics. In this context, ($T1$), ($T2$), and ($T4$) relate to temporal patterns as well as ranges and distributions to provide summarized information. All other tasks are related to graph-based data, and represent various attributes on either nodes ($T5$), or links ($T3$), ($T6$), ($T7$), ($T8$), and ($T9$).

5 Visualization and Interaction Design of TREESCOPE

Designing an interactive visualization tool that highlights the required details in the data, and yet has a low perceptual complexity, requires careful attention. Even within the requirements of TREESCOPE described in Section 4.1, there existed various degrees of freedom. A key feature of our design process was continued engagement with domain experts, which allowed us make many design decision considering both domain- and visualization-specific concerns. In order to accommodate space restrictions, we are not able to discuss the various initial prototypes of TREESCOPE; we focus on only the final visualization and the most important design choices. Figure 3 gives an overview of TREESCOPE; this section describes the visual encoding of various data elements (Sections 5.1–5.3), and how TREESCOPE maps these visual encodings to the required tasks through interactive exploration (Sections 5.4–5.6).

5.1 Encoding the Fat-Tree Topology

The primary use case of TREESCOPE is to visualize network traffic: every link in the network must be displayed without occlusion. As is known from the visualization literature [Ber67, BETT98, HMM00, GFC04, GFC05], adjacency matrices make

an excellent choice for displaying large-scale graphs, because they transform dense connectivity into compact, symmetrical, and uncluttered visual elements, leading to a highly scalable visual encoding. Therefore, we choose a matrix-view layout to display network traffic. This section describes how we encode the dense connectivity of fat-tree topology into a set of adjacency matrices, which when juxtaposed carefully, give a complete view of the network. Network traffic patterns can then be visualized as a heat map on these matrices, thereby providing a powerful focus-plus-context visualization to investigate congestion.

In order to fully appreciate the challenges and considerations in the visual design, we describe the visual encoding of TREESCOPE using an example of a full-scale production system containing 1296 compute nodes, 72 switches at L1 and L2 each, and 36 switches at L3, with each switch having 36 ports. The network consists of four pods, each containing 18 L1 and 18 L2 switches, and these pods are bridged together by two bundles of 18 L3 switches each.

Since pods represent both logical and structural units of networks, we encode the fat-tree network with respect to pods. This discussion exemplifies only a single pod, referring to Figure 4. Recall that each L1 switch in a given pod p is connected to multiple L0 nodes, all associated with the same pod; we denote their connectivity as $\{\mathrm{L0}\}_p \Rightarrow \{\mathrm{L1}\}_p$. Furthermore, the bipartite graph between the sets of L1 and L2 switches within the pod p can be denoted as $\{\mathrm{L1}\}_p \Leftrightarrow \{\mathrm{L2}\}_p$. Finally, each L2 switch is connected to an L3 switch from some bundle b; therefore, we qualify sets of L2 switches in a pod, i.e., $\{\mathrm{L2}\}_p$, further based on the bundle they connect to, i.e., $\{\mathrm{L2}\}_p^b$, and denote their connectivity to L3 switches as $\{\mathrm{L2}\}_p^b \Leftrightarrow \{\mathrm{L3}\}^b$. Therefore, the entire connectivity relevant to a given pod p can be summarized as $\{\{\mathrm{L0}\}_p \Rightarrow \{\mathrm{L1}\}_p \Leftrightarrow \{\mathrm{L2}\}_p^b \Leftrightarrow \{\mathrm{L3}\}^b\}$ for all b.

Since L0 nodes are connected to L1 nodes through dedicated links, which rules out interference on $\{\mathrm{L0}\}_p \Rightarrow \{\mathrm{L1}\}_p$ links, making them of less interest to the users. Therefore, in order to create a compactly packed visualization, TREESCOPE omits direct visualization of L0–L1 connectivity; the traffic on such links can be visualized indirectly, as will be shown later.

In general, a single $\{\mathrm{L1}\}_p \Leftrightarrow \{\mathrm{L2}\}_p$ interconnect can be represented using a single adjacency matrix, $\{\mathrm{L1}\}_p \times \{\mathrm{L2}\}_p$. To depict bidirectionality of links, we extend the matrix horizontally by duplicating L2 switches: one adjacency submatrix visualizes the traffic from $\{\mathrm{L1}\}_p$ to $\{\mathrm{L2}\}_p$, and the other shows the traffic in the opposite direction. For example, the figure shows directional submatrices in different colors (note the difference in the ordering of direction-specific submatrices, to be explained ahead).

To show two levels of interconnects, $\{\mathrm{L1}\}_p \times \{\mathrm{L2}\}_p^b \times \{\mathrm{L3}\}^b$, we stack the corresponding adjacency matrices on top of each other, such that the $\{\mathrm{L2}\}_p$ switches are shared horizontally between them. To suborganize the visualization with respect to bundles, we separate the $\{\mathrm{L2}\}_p$ and $\{\mathrm{L3}\}_p$ switches into subsets corresponding to bundles (two bundles in our example), and depict pod p as two juxtaposed sets of $\{\mathrm{L1}\}_p \times \{\mathrm{L2}\}_p^b \times \{\mathrm{L3}\}^b$ matrices. In this representation, the $\{\mathrm{L2}\}_p^b$ and $\{\mathrm{L3}\}^b$ are visualized as two separate stacks, horizontal and vertical, respectively; however, there is only a single vertical stack of $\{\mathrm{L1}\}_p$, since they do not depend upon b.

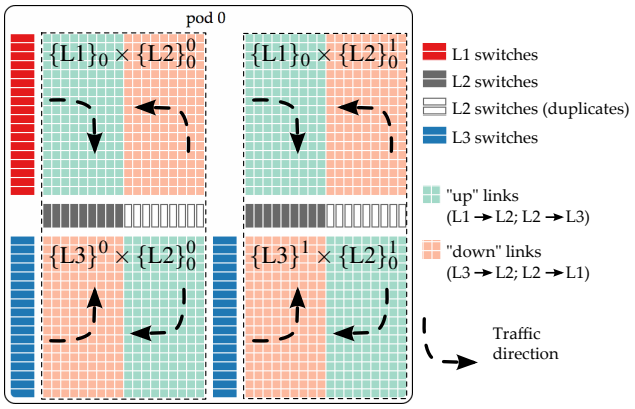

Figure 4: TREESCOPE *encodes large-scale graphs implied by the fat-tree topology using matrix-based representations. The figure shows one of the many pods (superscripted 0) and two bundles (subscripted 0 and 1) in the network. The connectivity within a given pod is visualized using a set of adjacency matrices between nodes at adjacent levels, i.e.,* $\{L1\}_0 \times \{L2\}_0^0$, $\{L3\}^0 \times \{L2\}_0^0$, *etc. The key features of the encoding include (1) omission of the less important,* $\{L0\}_0 \Rightarrow \{L1\}_0$, *connectivity, (2) hierarchical view of the pod by splitting the pod with respect to bundles (within dashed boxes), (3) separate matrices for "up" and "down" traffic (different colors), (4) duplication of L2 switches (solid vs. hollow gray switches) to enable visualization of directional traffic, and (5) reordering of directional submatrices to allow consistent aggregation (incoming or outgoing) of traffic for L2 switches.*

Finally, an important decision was to reorder the direction-specific submatrices. Note that the ordering of submatrices corresponding to "up" and "down" links is inverted for the $\{\mathrm{L3}\}^b \times \{\mathrm{L2}\}_p^b$ matrix, as compared to the $\{\mathrm{L1}\}_p \times \{\mathrm{L2}\}_p^b$. This modified ordering assigns a consistent traffic direction to every column with respect to $\{\mathrm{L2}\}_p^b$ switches. In other words, every column either brings traffic in or takes it out of a given L2 switch, allowing the visualization of a meaningful aggregate on them: "incoming" or "outgoing" traffic. Whereas the layout duplicates the L2 switches corresponding to the two traffic directions, duplicating L1 and L3 switches would require additional horizontal space. Therefore, we instead split each L1 and L3 switch in half to display corresponding "incoming" and "outgoing" traffic.

Our encoding not only arranges the network with respect to pods but also with respect to bundles within each pod, thus providing a hierarchical view of pods. Domain experts found this design easy to understand as it provides a physical layout perception and closely matched their intuition about the the network. Finally, this design is highly scalable, since all such bundles used in a network are typically of the same size, which leads to an optimal use of screen space as these adjacency matrices are of the same size, and can be packed compactly inside a pod. Note that the example illustrated in the figure contains only two bundles, and therefore, the pod contains two sub-parts. However, a larger number of submatrices can also be arranged horizontally, but still maintain the compactness and symmetry of the layout.

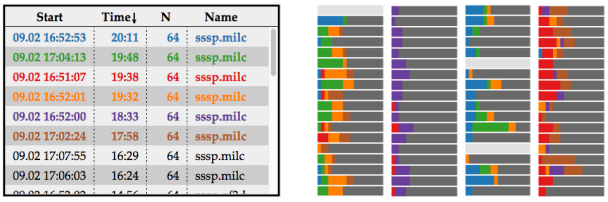

Figure 5: *Job-placement schemes can be visualized on L1 switches by color-coding them in proportion to the number of compute nodes occupied by corresponding jobs to the total number of compute nodes connected to them (the L1 switches). For selected jobs in the table, a stacked-histogram type visualization conveys which L1 switches contain compute nodes corresponding to different jobs.*

5.2 Encoding of Job Placement

The exploration of application jobs is a crucial component of our target analysis. Recall that the fat-tree view does not show L0 (compute) nodes. Therefore, in order to understand the impact of a given job-placement scheme, TREESCOPE instead uses an indirect visualization of job-to-node mapping: we show *job-to-L1 mapping*, which is a many-to-many mapping, and can be computed using job-to-node and node-to-L1 mappings, both of which are unique.

L1 nodes are visualized as horizontal rectangles (see Figure 4), which allows augmenting the visualization with job-placement information without any additional visual elements. As shown in Figure 5, TREESCOPE displays the job-to-L1 mapping using a stacked-histogram-type visualization, where "stacks" represent portions of L1 nodes occupied by different jobs: given a job that occupies n_j out of n active L0 nodes connected to a given L1 node, the width of the corresponding stack is $w \cdot n_j/n$, with w being the width of the rectangle representing the L1 node. The portion of an L1 node not used for any of the selected jobs is shown in dark gray.

5.3 Encoding of Quantitative and Categorical Data

Designing an effective color scheme is crucial for perceptual accuracy [SSM11, ZH16]. The color schemes used in TREESCOPE are inspired by ColorBrewer [HB03, Bre17]. TREESCOPE uses a single-hue color map to display the network traffic because such color maps are perceptually uniform [LH92, ZH16]. In some cases, the users may be interested in comparing the traffic to some reasonable quantity, for which TREESCOPE also offers choices of diverging color maps. TREESCOPE allows the user to choose up to four (single-hue and/or diverging) color maps: one each for "up"/"down" links, "incoming"/"outgoing" switch traffic. The default choices are single-hue color maps: 'green', 'orange', 'purple', and 'magenta', respectively. 'Red' is not used to avoid the red/green colorblindness issue. Fewer color maps may be used, which is strongly recommended to reduce chromatic complexity. Two categorical color maps are chosen to visualize summary plots and job placements. A paired scheme (light/dark pairs) is used for the former to maintain context between similar groups, e.g., L1 $\rightarrow$ L2, and L2 $\rightarrow$ L1 links. On the other hand, job placement uses a set of darker hues. Unfortunately, it is not possible to make these two schemes mutually exclusive due to a limited color set (considering color blindness). However, since these colored components are spatially distant on the screen, and do not interact with each other, the users did not have any problems in practice.

5.4 Exploration of Network Traffic

Upon selection of a metric of interest, such as "data_sent", through a UI dropdown, the user can explore the network behavior by interacting with the visualization as described below.

($T1$) **Temporal statistics.** The exploration typically begins with looking at statistics for arbitrary time ranges. TREESCOPE provides a time chart (see (b) in Figure 3) that shows the maximum or the average traffic (based on UI choice) for all links in the network for each time step for the ***entire time range***. This gives a quick overview and helps determine when the network utilization was high/low. The time chart is augmented with a *brush* that allows the user to select the ***active time range***, i.e., the time range to focus on.

($T2$) **Distributional statistics.** Next, users are typically interested in looking at the distribution of traffic counters to understand how many (if any) links were underutilized or overloaded. To this end, TREESCOPE displays a histogram (see (b) in Figure 3) of total traffic on every link, restricted to the active time range. The histogram also contains a brush that allows the user to select the ***active traffic range***, suggesting that the users are interested only in the links with a certain range of utilization. The UI also enables the user to view the complement of the brush selection to focus on both the low and the high end of the traffic distribution.

($T3$) **Combined/filtered statistics.** Through the UI, the user can choose to visualize the statistical plots with respect to the directionality and/or level of links. When one or both of these options are selected through the UI, the summary is decomposed into up to six groups (three levels and two directions): the time chart contains as many line plots, and the histogram is converted into a stacked histogram with as many stacks, which allows various types of comparisons, e.g., intra-pod vs. inter-pod traffic.

($T4$) **Fat-tree network visualization.** The core component of TREESCOPE is the detailed fat-tree network view, which shows traffic on every switch and link. The traffic metric under exploration is aggregated (summed) for every link in the network for the active time range, and displayed as a heat map on adjacency matrices. In the case of switches, the maximum traffic is visualized. Whereas the aggregation of traffic is performed for the active time range, the visibility of links is decided based on the active traffic range: links are displayed only if their traffic is in the range chosen by the user. Our choices are guided by the domain experts' interest in observing the total traffic flowing in the network for the active time range for underutilized and/or overloaded links. The aggregation and filtering are interactive, and the visualization updates dynamically with changes in user selection.

5.5 Exploration of Job Execution

A selectable and sortable table in the UI provides a listing of the available jobs. Users are often not interested in jobs running either for a small amount of time, or on a small number of nodes. Therefore, TREESCOPE provides UI options to filter jobs based on user-defined thresholds. Furthermore, an additional checkbox in the UI enables filtering of jobs based on time range: any jobs not running for any part of the active time range are not shown.

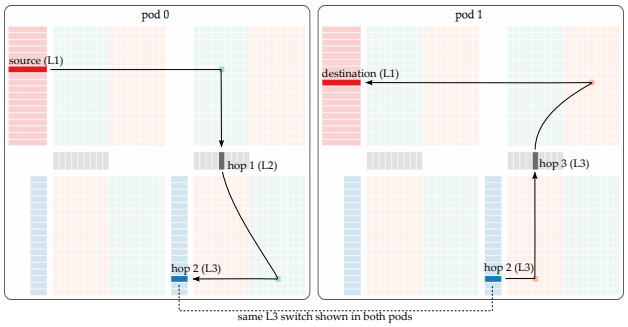

Figure 6: *The route between a pair of end nodes (or switches) can be visualized by highlighting the switches and links that the corresponding traffic goes through. The arrows denote the direction of routes, and show how the traffic is routed. Since the L3 switches are repeated in every pod, the traffic goes into a particular L3 switch but may come out from the same switch in a different pod.*

($T5$) **Visualization of job placement.** Through the UI, users can change the display mode of switches from "traffic" to "job mapping", revealing the stacked-histogram visualization of job placement. The "stacks" in the visualization correspond to the tabulation of job information through color. Any L1 nodes not occupied by selected jobs are faded away. As shown in Figure 5, when viewed in conjunction with this table, a simple and intuitive visualization of job placement in the network is obtained.

($T6$) **Visualization of traffic during a job's execution.** All components of the visualization are dynamic and interlinked. By selecting a job in the table, the user expresses interest in the network traffic only while the corresponding job as well as all other concurrent jobs were running. Upon selection of a job through the table, the active time range is changed to the time range of the job's execution, which, in turn, updates the entire visualization, including the traffic visualization in the fat-tree view. Moreover, if the time-based filtering of job table is enabled, the update of active time range in turn updates the table to contain concurrent jobs only.

5.6 Exploration of Traffic Routing

The ability to dynamically highlight routing information on the network enables the users to explore possible causes of network congestion. Given the encoding of fat-tree topology described in Section 5.1, the route between a pair of L1 switches in different pods is illustrated in Figure 6. The figure is augmented by arrows to help the reader trace the route; note that the arrows are not shown in TREESCOPE, because in real use cases, multiple such routes have to be visualized simultaneously, in which case such line visualizations create clutter and perceptual complexity. In particular, TREESCOPE visualizes routes by highlighting only the corresponding links and switches by fading out (using opacity) all others. Since we prefer to use quantitative single-hue color maps to display traffic on links, a white-colored link implies low traffic. In order to use opacity as a descriptive visual channel, we must, therefore, use a non-white background, which allows distinguishing a non-faded link with low traffic from a faded link. We next describe the route-based queries enabled by TREESCOPE.

($T7$) **Footprint of a (set of) job(s).** A selection in the job table implies the user's intent to explore how traffic corresponding to the selected job was routed through the network. Given a (set of) selected job(s), $\mathcal{J}$, TREESCOPE determines the set of switches and links used by $\mathcal{J}$. In particular, since main interest is in the bottlenecks created due to job interference, only the links used by *all* jobs in $\mathcal{J}$ are shown. Furthermore, since the goal is to understand which switches could be sending the corresponding traffic, all the switches used by *any* of the selected jobs are displayed.

($T8$) **Footprint of a (set of) network component(s).** The user can select one or more switches/ports in the network as *sources* (click) or *destinations* (shift+click), in which case, TREESCOPE highlights the route between the selected end points, determined using the active routing table.

($T9$) **Combinations of the above.** Common use cases combine both types of selection: the job-based route is treated as primary, and network selection is used to refine it into the final set of switches and links to be visualized. For example, in a typical exploration, a user would select a job, $\mathcal{J}$, and then possibly select an overloaded port, ℓ (say, as a source). Before the second selection, all L0 nodes of $\mathcal{J}$, $\mathcal{N}(\mathcal{J})$, are treated as both sources and destinations, and all possible routes available to the job are visualized. Upon the second selection, ℓ is treated as the source, and all possible routes through ℓ that could carry traffic to $\mathcal{N}(\mathcal{J})$ are computed. Specifically, if $d(\ell)$ is the set of destinations that ℓ can forward any incoming traffic to, then $d(\ell) \cap \mathcal{N}(\mathcal{J})$ is used as the set of destinations for computing routes. Through these interactive queries, the user can investigate causes of congestion, e.g., due to job interference, as discussed in Section 6.2.

6 Case Studies

The HPC domain experts, who have evaluated TREESCOPE, include performance analysis and optimization experts, network researchers, and system administrators at LLNL. They are convinced that TREESCOPE provides novel capabilities for intuitive and rapid analysis of performance monitoring data gathered on HPC networks. System administrators can use TREESCOPE to monitor the health of a supercomputer network via the summary overviews. The summary views can also aid in visually identifying time ranges of interest, which can then be loaded in the detailed fat-tree view. Subsequently, one can identify network hotspots or links with a high number of error counters and also the switches and/or jobs responsible for the anomalies. At the same time, TREESCOPE is a powerful tool to compare the impact of different workloads, routing schemes, or job placements on network congestion using data obtained either empirically or via simulations. Here, we present two studies performed by domain experts, who are also co-authors of this paper, to exemplify the aforementioned utilities of TREESCOPE. These co-authors work with the application developers and system administrators on performance analysis and network optimization of scientific workloads at LLNL. They have extensive experience in network analysis of large scale parallel applications, and have worked with other visualization tools designed for network analysis.

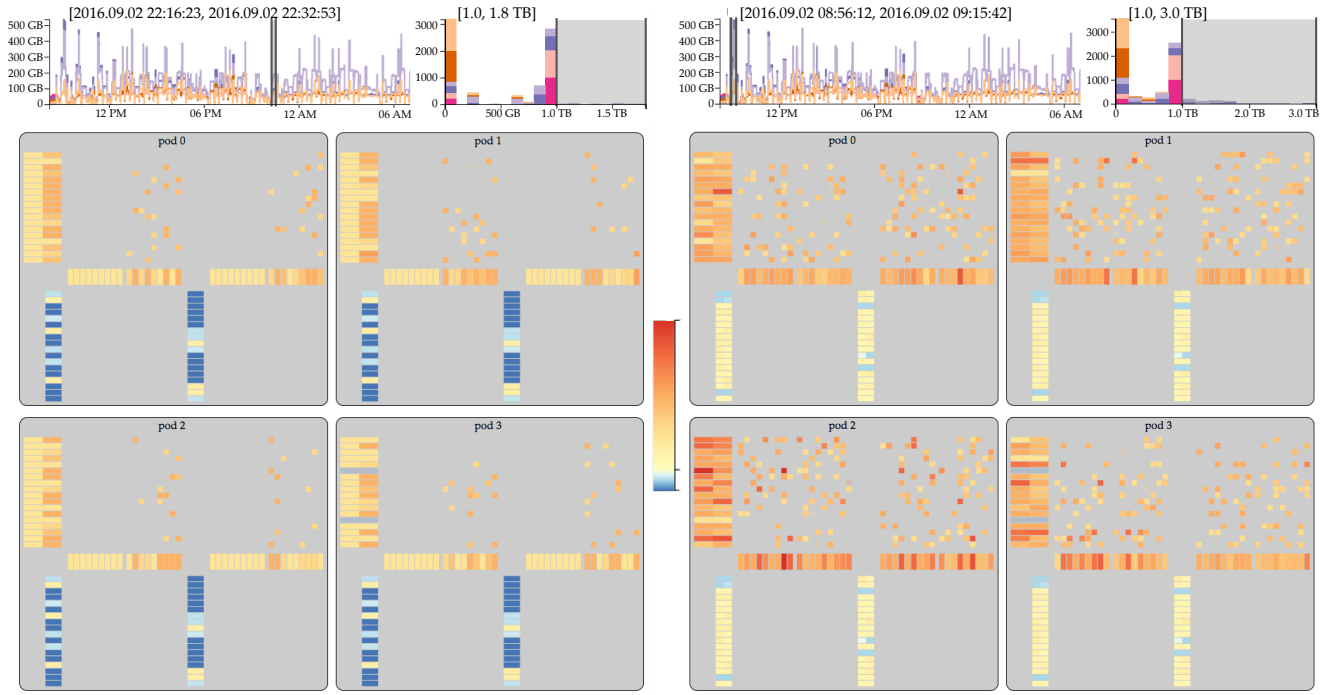

Figure 7: *Interactive filtering in* TREESCOPE *allows the user to visualize only the hotspot links (using histogram selection) to refine the understanding of network congestion. It is noted that whereas* ftree *routing (left) results in many links with above-average traffic (Figure 1), the number of hotspots links is significantly smaller in comparison to* SAR *routing (right). This explains the 17% better performance obtained with* ftree *routing for the selected application. The figure uses a diverging colormap for traffic, with yellow mapped to the average value.*

6.1 Comparison of Routing Schemes

The first case study analyzes experimental data gathered to compare different routing schemes for applications running on a 1296-node fat-tree cluster. A system reservation was requested for 24 hours and two routing schemes, denoted hereon as *ftree* and *SAR*, respectively, were tested for 12 hours each. *ftree* is a static routing [Zah10] used on many fat-tree based supercomputers. *SAR*, on the other hand, is a scheduling-aware routing [DH16], which attempts to distribute traffic over as many links as possible while considering the jobs running in the queue. For the 24-hour period, network counters were recorded along with job queue logs.

From the job queue logs, the domain experts identified two jobs, *ftree.qball* and *sar.qball*, that were running the same application (*qball*) with different routings and which had a significant difference in performance. The goal was to investigate how changes in network traffic due to routing were impacting the application's performance. Using two instances of TREESCOPE, the experts loaded the data for the two routing schemes and selected the two specific jobs. Once a job is selected, only counter data in the corresponding time frame is displayed. Since only a single job utilizing most of the machine (1024 nodes) is under consideration, all the traffic during that time is due to the selected job.

The traffic generated by the application using the two routing schemes is shown in Figure 1. The average traffic on a link is almost the same (< 0.2 GB) for both schemes, as expected, because the same application is sending traffic over the system. However, the observed maximum traffic is significantly higher for *SAR* (2.93 TB) as compared to *ftree* (1.78 TB). This can be seen in the respective histograms on the right. To study network traffic, we use a consistent diverging color map for the range 0–2.93 TB, with its mid point (yellow) mapped to the average traffic, and the maximum and minimum mapped to red and blue respectively. The experts noticed that *ftree* results in more L1 ↔ L2 links with higher than average traffic. Also notice that *ftree* uses a few specific L3 switches more heavily than others. In contrast, *SAR* distributes traffic over more L3 switches. For L1 ↔ L2 links, *SAR* results in an uneven distribution of traffic and fewer links with higher than average traffic (in comparison to *ftree*).

In order to identify the cause of the performance difference in the two executions (about 17%), the experts filtered the traffic to look at hotspots links, i.e., links with traffic higher than 1 TB. As shown in Figure 7, it was observed that *SAR* results in significantly more hotspots as compared to *ftree* (at all levels), denoted by many more L1 ↔ L2 links that are colored orange and red. This higher network traffic is the cause for congestion, and therefore the performance degradation with *SAR*. The experts also concluded that even for the traffic-oblivious *SAR*, some adversarial traffic patterns exist that slowed down the application when optimizing for intra-job all-to-all communication.

What distinguishes such an exploration from a typical otherwise-possible analysis is that TREESCOPE provides an easily understandable spatial context to the observed numbers. The domain experts can not only see the differences in distribution of traffic, but also find out where such links exist in the network and

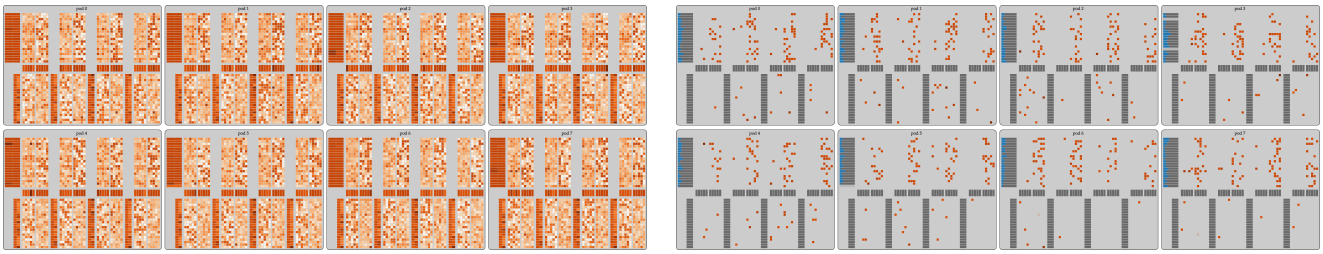

Figure 8: *Network traffic during the execution of several jobs on a 8-pod cluster is mapped to a white–orange color scale. Left: The overall traffic is high, yet no perceivable traffic patterns are observed. Right: Restricting the visualization to congested links and the traffic from a particular job,* Qbox*, highlights the connection between the two, suggesting the culpability of* Qbox *for creating congestion in all pods.*

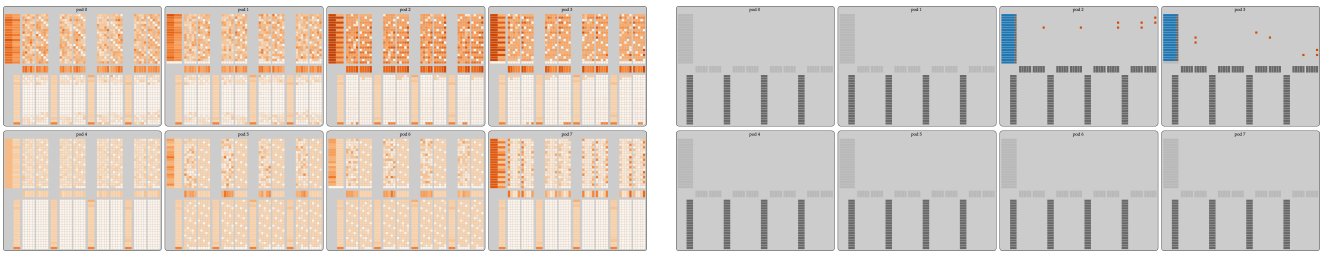

Figure 9: *The same experiment (compare with Figure 8), when performed with a different job-placement policy, shows that much fewer links are overloaded (left), suggesting less network congestion overall. Right: Visualizing the job-placement and traffic for the same job (*Qbox*), which is now mapped to pods 2 and 3 only, creates fewer hotspots, restricted locally to the corresponding pods, thus avoiding reducing congestion due to job interference.*

how they are connected to the rest of the network. For example, it becomes immediately obvious from the figure that even though the *ftree* routing distributes the incoming traffic from compute nodes (L0 $\rightarrow$ L1 $\rightarrow$ L2) very well and no hotspots are observed on the corresponding links, the traffic distribution going "down", i.e., on L2 $\rightarrow$ L1 links, is not well balanced. Such insights are highly valuable to the domain experts to further tune the routing algorithms or job-placements for a supercomputer.

6.2 Optimizing the Performance of Applications

In this section, we demonstrate the utility of TREESCOPE for investigating degraded network performance and finding solutions that improve overall application performance on production HPC systems. On such systems, simultaneous execution of multiple jobs that share network resources often results in inter-job interference. Even within a job, different MPI processes may contend with each other for network bandwidth. These factors result in congestion and negatively impact the performance of individual jobs. Here, the goal is to help the domain experts identify root causes of network congestion in multi-job environments.

Network simulation tools such as TraceR [JBW*16] are used to study congestion scenarios because they provide a high degree of configurability to interconnect experts that is typically not possible on production systems. Using TREESCOPE, the domain experts explore the data obtained from one such simulation, where multiple large jobs are concurrently simulated on a 3200-node, 8-pod prototype system built using 40-port switches and 400-node pods. As is typical in a production system, nodes are assigned to jobs in a fragmented fashion throughout the system, i.e., a typical job is allocated nodes attached to several switches and pods.

Figure 8 (left) visualizes the traffic distribution for the entire system due to all jobs, and highlights the presence of generally high congestion (darker colors) without any specifically identifiable patterns. Thus, despite observing congestion, it remains difficult to correlate it with potential root causes, such as a particular job, placement strategy, or communication pattern. In such scenarios, TREESCOPE's ability to select individual job(s) and show only its footprint on the network, in combination with value-based filtering of links, provides a useful functionality for exploring the traffic distribution. By viewing the traffic hotspots created by different jobs one by one, the domain experts were able to isolate a single job, *Qbox*, which creates most of the heavily loaded links observed in Figure 8 (right). The job placement for *Qbox*, highlighted in blue on L1 nodes, shows that it was allotted a large number of nodes across all available pods. For all other jobs, TREESCOPE showed that a very small fraction of links were hotspot links, implying that *Qbox* is the main cause of congestion on network in this workload, and may negatively impact the performance of other jobs.

To isolate the traffic initiated by *Qbox*, the domain experts devised a *pod-based* job-placement policy, where jobs are (preferably) allocated to nodes belonging to same pods. *Qbox* was assigned two dedicated pods, pods 2 and 3. Figure 9 (left) shows that for the pod-based placement policy, the traffic distribution in different pods changes based on the job allocated to them. Indeed, pods 2 and 3 show high traffic and non-uniform distributions, whereas other pods, especially pods 4 and 7, show light traffic and uniform distributions. TREESCOPE makes it easy to confirm, through job-based and value-based filtering, and through visualization of job-placement, see Figure 9 (right), that almost all of the congestion is indeed created by *Qbox*.

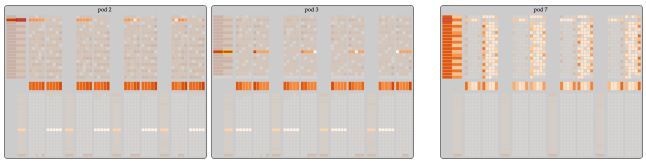

Figure 10: *Visualizing links that can be used for communication from a switch or between a pair of switches show that task-to-node mapping and routing policy result in overloading some links while underutilizing other links. Traffic for two different jobs is shown.*

Thus, as a result of the exploration enabled by TREESCOPE and redesigning job-placement scheme, the performance of individual jobs improves by 20.8% to 42.6%.

The isolation of job-specific traffic with a pod-based node allocation policy also provides an opportunity for domain experts to study link-usage patterns of individual jobs in order to optimize their performance. Figure 10 demonstrates that viewing traffic on all links available for communication to individual switches can identify causes of congestion. The left image shows that when a pair of switches is selected as a source-destination pair for *Qbox*, some switches are overloaded whereas others are underutilized. The image at the right shows that for a different job, when a switch is selected as a source for communication with all other nodes used for the same job, imbalance among switch utilization is present. This suggests that an incompatibility between communication patterns of these jobs and the static routing policy is the cause of network congestion. These findings, which are hard to make without the features provided by TREESCOPE, can help domain experts devise solutions that can improve performance.

7 Conclusion

This paper introduces TREESCOPE, an interactive web-based visualization tool for exploring traffic (and other relevant hardware counters) and investigating the effects of job placement and routing schemes on supercomputers using fat-tree networks. Using a new matrix-based representation of the fat-tree topology, TREESCOPE combines various sources of data, and supports a comprehensive exploration through various types of complex queries. TREESCOPE presents high-level and summarized, as well as detailed, per-link and per-switch information to the user in a compact manner. Some of the target users of TREESCOPE, both network researchers and system administrators, have been actively involved in its design; we summarize our collaborative process, including some important decisions that led to the final visual design of TREESCOPE.

TREESCOPE enables users to not only identify hotspots in the network, but also focus on the applications responsible for creating the congestion. Using a combination of data (network counters, job queue logs, job placements, and routing schemes), we demonstrate the effectiveness of TREESCOPE in two case studies — one for the traffic observed on real production systems, and another for a simulated network study to design better job placement schemes. In both scenarios, various features in TREESCOPE, such as traffic visualization, visualization of job placement, and route-finding, have proven useful allowing users to draw new insights.

Through evaluation of the current version of TREESCOPE, the domain experts at LLNL have found it immensely useful for a wide variety of domain queries. Several new directions have also been identified to further improve the applicability of the tool. Going forward, we plan to extend TREESCOPE to support more-detailed data, such as the communication graph of processes in an application, which will expose a new level of detail in the analysis. Although the primary focus of TREESCOPE is on commonly used three-level fat-tree topology for HPC clusters, several aspects of the design are more generalizable than currently supported in TREESCOPE. For example, the same visual design could support other types of interaction and queries for the administrators in data centers. The current visual design can also be enhanced to support other configurations of fat-tree topologies, such as beyond three-level fat-trees and dual-plane connections. Finally, similar visualizations have been utilized in other related contexts, such as inter-domain routing protocols [TRNC06]; we would like to explore new application domains for TREESCOPE.

TREESCOPE addresses a clear need in the HPC community to explore traffic on large-scale networks, as users are looking for new ways to explore network data. To expand the user base beyond our institution, TREESCOPE has been released publicly under BSD license: https://github.com/LLNL/TreeScope.

Acknowledgements

We would like to thank Kevin Brown (Tokyo Institute of Technology) and Israa Al-Qassem (Purdue University) for help and ideas on the first prototype, as well as Todd Gamblin and Tim Meier (both at LLNL) for fruitful discussions and feedback on several prototypes of the tool. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL) under contract DE-AC52-07NA27344.

References

[ABC*05] ADIGA N. R., BLUMRICH M. A., CHEN D., COTEUS P., GARA A., GIAMPAPA M. E., HEIDELBERGER P., SINGH S., STEINMACHER-BUROW B. D., TAKKEN T., TSAO M., VRANAS P.: Blue Gene/L torus interconnection network. *IBM J. Res. Dev. 49*, 2 (2005), 265–276. 3

[ABHR*13] ALPER B., BACH B., HENRY RICHE N., ISENBERG T., FEKETE J.-D.: Weighted graph comparison techniques for brain connectivity analysis. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems* (2013), pp. 483–492. 3

[AES05] AMAR R. A., EAGAN J., STASKO J. T.: Low-level components of analytic activity in information visualization. In *Proc. of the IEEE Symp. on Information Visualization* (2005), pp. 111–117. 5

[AS05] AMAR R. A., STASKO J. T.: Knowledge precepts for design and evaluation of information visualizations. *IEEE Trans. on Vis. and Comp. Graph. 11*, 4 (2005), 432–442. 3

[Bar] BARCELONA SUPERCOMPUTING CENTER: *Paraver: a flexible performance analysis tool*. https://tools.bsc.es/paraver. 3

[BDM15] BROWN K. A., DOMKE J., MATSUOKA S.: Hardware-centric analysis of network performance for MPI applications. In *Proc. of the IEEE Int. Conf. on Parallel and Distributed Systems* (2015), pp. 692–699. 3

[Ber67] BERTIN J.: *Sémiologie graphique: Les diagrammes, les réseaux, les cartes*, editions de l'ecole des hautes etudes en sciences ed. Paris, France, 1967. 3, 5

[BETT98] BATTISTA G. D., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing: Algorithms for the Visualization of Graphs*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. 3, 5

[BJL*16] BHATELE A., JAIN N., LIVNAT Y., PASCUCCI V., BREMER P.-T.: Analyzing network health and congestion in dragonfly-based supercomputers. In *Proc. of the IEEE Int. Parallel & Distributed Processing Symp.* (2016), pp. 93–102. 2, 3, 4

[BM13] BREHMER M., MUNZNER T.: A multi-level typology of abstract visualization tasks. *IEEE Trans. on Vis. and Comp. Graph. 19*, 12 (2013), 2376–2385. 5

[BNTM16] BREHMER M., NG J., TATE K., MUNZNER T.: Matches, mismatches, and methods: Multiple-view workflows for energy portfolio analysis. *IEEE Trans. on Vis. and Comp. Graph. 22*, 1 (2016), 449–458. 3, 5

[Bre17] BREWER C. A.: ColorBrewer 2, Mar. 2017. http://www.colorbrewer2.org/. 7

[CDJM14] CHENG S., DE P., JIANG S. H.-C., MUELLER K.: TorusVisND: Unraveling high-dimensional torus networks for network traffic visualizations. In *Proc. of the 1st Work. on Visual Performance Analysis* (2014), pp. 9–16. 3

[Clo53] CLOS C.: A study of non-blocking switching networks. *The Bell System Technical Journal 32*, 2 (1953), 406–424. 2

[DH16] DOMKE J., HOEFLER T.: Scheduling-Aware Routing for Supercomputers. In *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (2016), SC '16, pp. 13:1–13:12. 9

[GFC04] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: A comparison of the readability of graphs using node-link and matrix-based representations. In *Proc. of the IEEE Symp. on Information Visualization* (2004), pp. 17–24. 3, 5

[GFC05] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization 4*, 2 (2005), 114–135. 3, 5

[HB03] HARROWER M., BREWER C. A.: Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal 40*, 1 (2003), 27–37. 7

[HCR01] HAYNES R., CROSSNO P., RUSSELL E.: A visualization tool for analyzing cluster performance data. In *Proc. of the IEEE Int. Conf. on Cluster Computing* (2001), pp. 295–302. 3

[HE91] HEATH M. T., ETHERIDGE J. A.: Visualizing the performance of parallel programs. *IEEE Software 8*, 5 (1991), 29–39. 3

[HF06] HENRY N., FEKETE J. D.: MatrixExplorer: a dual-representation system to explore social networks. *IEEE Trans. on Vis. and Comp. Graph. 12*, 5 (2006), 677–684. 3

[HMM00] HERMAN I., MELANCON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Trans. on Vis. and Comp. Graph. 6*, 1 (Jan 2000), 24–43. 3, 5

[HS12] HEER J., SHNEIDERMAN B.: Interactive dynamics for visual analysis. *Commun. ACM 55*, 4 (Apr. 2012), 45–54. 5

[IGJ*14] ISAACS K. E., GIMÉNEZ A., JUSUFI I., GAMBLIN T., BHATELE A., SCHULZ M., HAMANN B., BREMER P.-T.: State of the Art of Performance Visualization. In *EuroVis - STARs* (2014), Borgo R., Maciejewski R., Viola I., (Eds.), The Eurographics Association. 3

[JBW*16] JAIN N., BHATELE A., WHITE S., GAMBLIN T., KALE L. V.: Evaluating HPC networks via simulation of parallel workloads. In *Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis* (2016), SC '16, pp. 14:1–14:12. 2, 10

[Lei85] LEISERSON C. E.: Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. on Comp. C-34*, 10 (1985), 892–901. 2

[LH92] LEVKOWITZ H., HERMAN G. T.: Color scales for image data. *IEEE Comp. Graph. and App. 12*, 1 (1992), 72–80. 7

[LLB*12] LANDGE A. G., LEVINE J. A., BHATELE A., ISAACS K. E., GAMBLIN T., SCHULZ M., LANGER S. H., BREMER P.-T., PASCUCCI V.: Visualizing network traffic to understand the performance of massively parallel simulations. *IEEE Trans. on Vis. and Comp. Graph. 18*, 12 (2012), 2467–2476. 2, 3, 4

[LMR*17] LI J. K., MUBARAK M., ROSS R. B., CAROTHERS C. D., MA K.-L.: Visual analytics techniques for exploring the design space of large-scale high-radix networks. In *Proc. of the IEEE Int. Conf. on Cluster Computing (CLUSTER)* (2017), pp. 193–203. 2, 3

[LPP*06] LEE B., PLAISANT C., PARR C. S., FEKETE J.-D., HENRY N.: Task taxonomy for graph visualization. In *Proc. of the AVI Work. on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization* (2006), BELIV, pp. 1–5. 5

[MIB*14] MCCARTHY C. M., ISAACS K. E., BHATELE A., BREMER P.-T., HAMANN B.: Visualizing the five-dimensional torus network of the IBM Blue Gene/Q. In *Proc. of the 1st Work. on Visual Performance Analysis* (2014), pp. 24–27. 3

[MSDS] MEUER H., STROHMAIER E., DONGARRA J., SIMON H.: Top500 Supercomputer Sites. http://www.top500.org. 2

[Mun09] MUNZNER T.: A nested model for visualization design and validation. *IEEE Trans. on Vis. and Comp. Graph. 15*, 6 (2009), 921–928. 3, 4

[PCJ97] PURCHASE H. C., COHEN R. F., JAMES M. I.: An experimental study of the basis for graph drawing algorithms. *J. Exp. Algorithmics 2* (1997). 3

[SMM12] SEDLMAIR M., MEYER M., MUNZNER T.: Design study methodology: Reflections from the trenches and the stacks. *IEEE Trans. on Vis. and Comp. Graph. 18*, 12 (2012), 2431–2440. 3

[SMM*13] SIGOVAN C., MUELDER C., MA K.-L., COPE J., ISKRA K., ROSS R.: A visual network analysis method for large-scale parallel I/O systems. In *Proc. of the IEEE Int. Parallel & Distributed Processing Symp.* (2013), pp. 308–319. 3

[SSM11] SILVA S., SANTOS B. S., MADEIRA J.: Using color in visualization: A survey. *Comp. & Graph. 35*, 2 (2011), 320–333. 7

[TRNC06] TEOH S. T., RANJAN S., NUCCI A., CHUAH C.-N.: Bgp eye: A new visualization tool for real-time detection and analysis of bgp anomalies. In *Proc. of the 3rd Int. Work. on Vis. for Comp. Sec.* (2006), pp. 81–90. 11

[TSW14] THEISEN L., SHAH A., WOLF F.: Down to earth – how to visualize traffic on high-dimensional torus networks. In *Proc. of the 1st Work. on Visual Performance Analysis* (2014), pp. 17–23. 3

[WCC*17] WANG X., CHOU J.-K., CHEN W., GUAN H., CHEN W., TIANYI LAO, MA K.-L.: A visual analytics system for optimizing communications in massively parallel applications. In *Proc. of the IEEE Conf. on Visual Analytics Science and Technology* (2017). 3

[Wij06] WIJK J. J. V.: Bridging the gaps. *IEEE Computer Graphics and Applications 26*, 6 (Nov 2006), 6–9. 3

[XZWX17] XIA W., ZHAO P., WEN Y., XIE H.: A survey on data center networking (dcn): Infrastructure and operations. *IEEE Communications Surveys Tutorials 19*, 1 (2017), 640–656. 2

[Zah10] ZAHAVI E.: *D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees*. Tech. rep., Israel Institute of Technology, Aug. 2010. 9

[ZH16] ZHOU L., HANSEN C. D.: A survey of colormaps in visualization. *IEEE Trans. on Vis. and Comp. Graph. 22*, 8 (2016), 2051–2069. 7

[ZSC03] ZHOU C., SUMMERS K. L., CAUDELL T. P.: Graph visualization for the analysis of the structure and dynamics of extreme-scale supercomputers. In *Proc. of the ACM Symp. on Software Vis.* (2003), pp. 143–149. 3