Sparsifying Network Topologies for Application Guidance

Michael Scharf, Gordon Wilfong, Lisa Zhang*
Alcatel-Lucent Bell Labs
Email: michael.scharf@alcatel-lucent.com, gtw@research.bell-labs.com, ylz@research.bell-labs.com

***Abstract*—Topology managers expose network information to applications to improve application-level resource management. An example for various ongoing standardization activities is Application-Layer Traffic Optimization (ALTO). Due to privacy and security constraints, exposed network information has to be filtered according to policies. As part of such policies, distance information can be abstracted by presenting coarser distances over pairs of clustered nodes rather than the precise distances between all node pairs. We refer to this process as *distance sparsification*.**

The contribution of this paper is two-fold, the first being a new policy system to enforce abstraction. The second and the main contribution addresses the algorithmic challenge. For the latter, we consider two types of distance sparsification algorithms. The first variant takes as input a matrix of pairwise distances. The sparsification algorithm produces a smaller distance matrix by clustering the nodes into clusters. The second variant instead collapses an edge-weighted graph. We measure the performance of the algorithms by the accuracy of the resulting sparsified distances, and we show that matrix sparsification outperforms graph sparsification. We further observe the trade-off between the accuracy and the size of the sparsified representation. In addition, we also extend our algorithms to handle labeled data, i. e., abstraction policies explicitly mark a number of destinations as reference points. Such additional information can improve the distance sparsification.

I. Introduction

Insight into the network topology is important for many applications and uses of networks. Network-aware guidance can improve application-level resource management such as resource selection, placement of entities such as Content Delivery Network (CDN) caches, or calendaring of bandwidth in data center networks. In general, a topology manager is a key component of a Software-Defined Networking (SDN) architecture. As a result, there are various ongoing standardization activities for interfaces describing a network topology.

For instance, according to ref. [1], a topology manager could provide a "coherent picture of the network state". Application-based network operations [2] may bring together many existing technologies for gathering information about the resources available in a network. An standardized topology exposure interface is Application-Layer Traffic Optimization (ALTO) [3], which offers a "better-than-random" guidance for application resource selection decisions [4]. There are also various other ongoing standardization activities, e. g., for SDN.

A topology exposure system can either offer a *transparent* or an *abstract* view on the network. An example for the former category is a Traffic Engineering Database (TED) of a Path Computation Element (PCE), i. e., a data store of topology information enhanced with capability data such as bandwidth and active status information.

The second class of systems exposes a more abstract view, since network administrators are often reluctant to share operational network data because of privacy and security concerns. Also, applications using a topology manager typically only require insight into a subset of the network topology, e. g., the topology between instances of that application. In other words, the topology manager might filter the available data and it may hide or summarize parts that are irrelevant. This second category is in particular appropriate if the application querying the system is not a network management application, but, e.g., a CDN [4] or another network-external user. In this case an abstract representation of the network topology can be sufficient [5], [6]. This paper considers the latter class of topology exposure systems.

The topology abstraction involves two related problems. First, a network operator offering a topology exposure service will need a policy enforcement system to hide network topology details that should not be revealed for security or privacy reasons. And second, in particular in large and complex networks it makes sense to sparsify the network topology, e. g., by aggregating close subnetworks [7]. Such aggregation is one means to hide internal network structures. In addition, it also reduces the amount of data describing a topology and thus helps to ensure scalability for large topologies.

The contribution of this paper is two-fold. The first contribution in Section II is a new policy system to enforce topology abstraction. This addresses the first problem above. The second and main part of the paper studies the algorithmic challenge of distance sparsification. Section III defines two variants of abstraction algorithms, namely *graph sparsification* and *matrix sparsification*, which can be used inside the topology exposure system. In Section IV, we present and analyze heuristics for both variants. Both algorithms are closely related to clustering, for which we show NP-completeness among other properties of the proposed heuristics. As algorithmic contribution we propose a modular three-step framework for the matrix sparsification method. In Section V we compare numerically the sparsification quality of the proposed heuristics when applied to three network instances. We observe behaviors that are consistent with our earlier analysis. We further study the trade-off between the accuracy and the size of the sparsified representation. Finally, Section VI briefly surveys related work and Section VII concludes this paper.

* Author names are listed in alphabetical order.

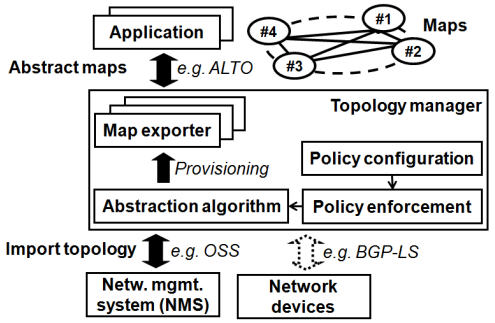

Fig. 1. Topology exposure system architecture

II. Proposed Topology Exposure System

A. System Architecture

Topology exposure systems typically have a client-server architecture. The server component is a topology manager that exposes network topology data to clients, i. e., applications requiring network awareness. An overview of this architecture is depicted in Fig. 1.

There are two classes of interfaces in this architecture: First, an interface between clients and the topology manager is needed. ALTO [3] is one example for a Representational State Transfer (REST) interface that offers both an abstract map view and a ranking service. Instead of ALTO, other interfaces could be used as well, e.g., using models defined in YANG [8]. Second, the topology manager has to learn the network topology. Given the complexity and heterogeneity of networks, a topology manager may have to correlate different input sources to get a holistic view [4]. Example data sources include Operations Support System (OSS) interfaces to Network Management Systems (NMS) [7], control plane protocols such as BGP-LS [9], or other SDN interfaces.

A topology exposure system can also be part of an SDN architecture that allows reconfiguration of the network. In this case, the topology manager may be part of the SDN controller or a dedicated component that provides PCE functionality [2]. This paper focuses on a system that provides read-only access to the topology information and therefore we do not further discuss SDN or PCE use cases.

B. Policy Enforcement

A key component of a topology manager is a policy system that governs what aspects of a network topology are actually

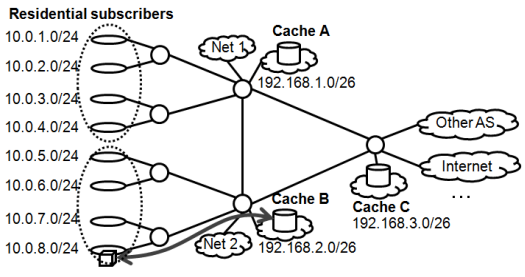

Fig. 2. Topology exposure example for a CDN

```
prefix-list "residential"
  prefix 10.0.0.0/8 longer
exit
prefix-list "cache_A"
  prefix 192.168.1.0/26 longer
exit
prefix-list "cache_B"
  prefix 192.168.2.0/26 longer
exit
prefix-list "cache_C"
  prefix 192.168.3.0/26 longer
exit
policy-statement "cdn"
  entry 1
    from
      protocol bgp
      prefix-list "cache_A"
    exit
    action accept
      reference "CacheA"
    exit
  exit
  entry 2
    from
      protocol bgp
      prefix-list "cache_B"
    exit
    action accept
      reference "CacheB"
    exit
  exit
  entry 3
    from
      protocol bgp
      prefix-list "cache_C"
    exit
    action accept
      reference "CacheC"
    exit
  exit
  entry 4
    from
      prefix-list "residential"
    exit
    action accept
    exit
  exit
  default-action reject
exit
```

Fig. 3. Example for the topology manager import policies

made available to a given user. These abstraction policies are likely to be specific to each user, i. e., each application using a topology manager may get an own view of the network. For instance, a CDN serving residential users as shown in Fig. 2 would likely be interested in knowing which cache out of A, B, and C is the topological best cache for any subscriber. That CDN may have no need to know about parts of the network not traversed by CDN traffic. The topology manager could use filter policies to remove parts of a network from the maps that are made available to various applications.

Reference [7] presents a general framework for a policy manager to derive application-specific maps. In the following, we propose a new policy solution to control the abstraction within such a framework. Filter policies have some similarity to import and export filters for the Routing Information Base (RIB) in a router, e. g., for the Border Gateway Protocol (BGP) [10]. Typical route policies contain policy statements with ordered entries that specify match conditions and accept/reject actions. The match criteria can be based on various parameters including the source/destination, routing protocol, prefix lists, or other properties such as Autonomous system (AS) path matches or BGP communities.

Our policy system reuses this principle. Filtering is achieved by a sequence of entries describing accept/reject actions. Routes in the topology obtained from data input sources are compared to those entries, including default ones. The topology manager does not expose map data for destinations matching a reject condition (`reject` action). Figure 3 lists an example for a policy definition corresponding to the example in Fig. 2, using the policy syntax defined in [11]. This policy definition ensures that the CDN will only learn topology data between caches and residential users, which are here given by prefix lists with explicit `accept` rules. Other matching constraints could be used as well [11].

The above policy solution for a topology manager also includes two new functions that differ from typical RIB import policies. First, we add support for explicit labeling of destinations in the network, for instance, by IP address ranges. In Fig. 3, this labeling is defined by the `reference` entries. By this policy the administrator can explicitly flag destinations in a network as potentially relevant for the client. In the CDN example, this new feature would particularly be useful for the locations of the caches, as further detailed later.

TABLE I. EXAMPLE ABSTRACT MAP (SIMPLIFIED)

	CacheA	CacheB	CacheC
Residential1	3 hops	4 hops	4 hops
Residential2	4 hops	3 hops	4 hops

The second feature that differs from typical route policies is support for automatic aggregation of destinations with similar but not necessarily identical costs. One can observe in Fig. 2 that a CDN load balancer, which maps residential subscribers to the closest cache, does not have to distinguish between all individual subnets; instead they could also be aggregated into two clusters. All subnets inside one cluster have the same closest cache. Determining these clusters and the distances between those clusters requires abstraction algorithms. Corresponding sparsification algorithms are introduced and studied in the following sections.

An expected example outcome for a map resulting from this policy configuration as well as follow-up distance sparsification can be found in Table I. In this example, we use the hop count as distance metric, but other available distance metrics could be used as well in the following sections (e. g., geographical distance, delay).

III. TWO VARIANTS OF DISTANCE SPARSIFICATION

A. Matrix vs. Graph Sparsification

In order to generate this representation we study two variants of distance sparsification, distance matrix sparsification (MATRIXSPAR) and graph sparsification (GRAPHSPAR). The former takes as input a set V of n nodes and an $n \times n$ matrix M where the (u, v)-entry is the distance $d(u, v)$ between nodes u and $v \in V$. From such input and a target size $k < n$, a sparsification algorithm partitions V into k clusters $C_1, C_2, \ldots, C_k$ and defines a $k \times k$ *sparsified distance matrix* M^s whose (i, j)-entry is the approximated distance $d^s(u, v)$ between $u \in C_i$ and $v \in C_j$.

GRAPHSPAR takes as input a graph $G = (V, E)$ for which each edge has an associated length. We use $d(u, v)$ to represent the shortest-path distance in G between nodes $u, v \in V$. A sparsification algorithm partitions the nodes in V into $k < n$ cluster nodes $V^s = \{C_1, C_2, \ldots, C_k\}$, and outputs a smaller graph $G^s = (V^s, E^s)$ where $|V^s| < |V|$ and $|E^s| < |E|$. The shortest-path distance in G^s between the cluster nodes $C_i, C_j \in V^s$ is the approximated distance $d^s(u, v)$ between $u \in C_i$ and $v \in C_j$. Note that while throughout the paper we assume that distances are symmetric, i.e., $d(u, v) = d(v, u)$ for all $u, v \in V$, our work is applicable to asymmetric distances.

B. Measures for Accuracy

We measure the performance of the sparsification algorithms by the accuracy of the resulting sparsified distances, i.e. we would like $d^s(u, v)$ to be as close to $d(u, v)$ as possible. For the most of the paper we examine the *mean square error* (MSE)

$$MSE = \frac{1}{|V|^2} \sum_{u,v \in V} \delta(u, v), \tag{1}$$

where

$$\delta(u, v) = (d(u, v) - d^s(u, v))^2, \tag{2}$$

is the distance error squared for node pair u and v. Additional metrics motivated by use cases will be considered in Section V-C.

Let us take a look at how GRAPHSPAR and MATRIXSPAR are related with respect to the MSE measure. Consider a GRAPHSPAR instance on input graph G and let $OptMSE(G)$ be the best possible MSE for sparsifying graph G. We also define a comparable MATRIXSPAR instance on distance matrix M where the matrix entries represent the shortest-path distance on G. Let $OptMSE(M)$ be the best possible MSE for sparsifying matrix M. We have

Theorem III.1. *An optimal solution for* MATRIXSPAR *necessarily performs at least as well as* GRAPHSPAR*, namely*

$$OptMSE(G) \geq OptMSE(M).$$

The above inequality holds for any metric, not just MSE.

Proof: Let $OptG^s$ be the optimal sparsified graph under the optimal algorithm for GRAPHSPAR, and this graph results in MSE equal to $OptMSE(G)$. Let X be the $k \times k$ distance matrix where each entry of X defines the shortest-path distance on graph $OptG^s$. Note that X is one candidate sparsified matrix to the MATRIXSPAR instance M, and X produces MSE equal to $OptMSE(G)$. Let M^s be the optimal sparsified matrix under the optimal algorithm for MATRIXSPAR. By definition, M^s is necessarily no worse than X and therefore $OptMSE(M)$ is no worse than $OptMSE(G)$.

On the other hand, note the M^s may not be realizable by any graph with edge lengths. First, M^s does not have to satisfy the triangle inequality and therefore does not have to define a metric space, whereas a distance matrix defined by the shortest-path distances on any graph has to. Second, GRAPHSPAR restricts the number of edges in the sparsified graph which may make some distance matrix not realizable. For example, if the distance matrix defines the distances on nodes u, v and w to be $d(u, v) = d(v, w) = d(w, u) = 1$, but if the graph is restricted to have 2 edges uv and vw only, then there is no way to realize $d(u, w) = 1$ in the graph. Therefore, MATRIXSPAR has more freedom in sparsification and necessarily preserves distances at least as well.

The above argument does not rely on specifics of MSE, and the inequality therefore holds for any sparsification metric. ■

While the above theorem addresses the optimal behavior for MATRIXSPAR and GRAPHSPAR, we will see in Section V that the heuristics for MATRIXSPAR proposed in Section IV-A and for GRAPHSPAR introduced in earlier publication [7] exhibit similar relative behavior.

IV. SPARSIFICATION ALGORITHMS

A. Distance Matrix Sparsification

Our heuristic for MATRIXSPAR partitions V into k clusters for a prespecified integer $k < n$ and computes a $k \times k$ matrix M^s for cluster-to-cluster distances. As a baseline, we offer a simple algorithm that consists of three modules. Each step can be realized in multiple ways, and we present and discuss some options.

1) Select k cluster centers.

2) Partition V into k clusters. This can be implemented as the standard *Voronoi diagram*. For $u \in V$, it is assigned to the closest center, namely $argmin_{\ell \in L} d(u, \ell)$. Let $\{C_1, C_2, \ldots, C_k\}$ be the resulting clusters.
3) Compute cluster-to-clusters distances. We define these aggregate distances to be the average of node-to-node distances. In particular,

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{u \in C_i, v \in C_j} d(u, v) \quad (3)$$

The above defines the intra-cluster distance within cluster C_i when $i = j$ and the inter-distance between clusters C_i and C_j when $i \neq j$.

The first step of cluster center selection and the second step of partitioning can utilize many known algorithms (see Section VI). As a starting point we focus on the simple and intuitive methods of k-center and Voronoi diagram. The third step of our modular algorithmic framework is in fact optimal for the mean square error as our objective function:

Theorem IV.1. *For any given k clusters, the cluster-to-cluster distance defined in Eq. (3) result in the following properties.*

1) *The total distance before and after sparsification is preserved, i.e. $\sum_{u,v \in V} d(u, v) = \sum_{u,v \in V} d^s(u, v)$.*
2) *For any given partition of k clusters, the cluster-to-cluster distance defined in Eq. (3) minimizes MSE.*

Proof: Consider any pair of clusters C_i and C_j where C_i can be the same as C_j. For item 1), We have

$$\sum_{u \in C_i, v \in C_j} d^s(u, v) = |C_i||C_j| d(C_i, C_j) = \sum_{u \in C_i, v \in C_j} d(u, v)$$

by Eq. (3). Summing over all cluster pairs proves item 1).

For item 2), let x be the cluster-to-cluster distance between C_i and C_j. The total square error incurred from pairs (u, v) where $u \in C_i$ and $v \in C_j$ is

$$\begin{aligned} & \sum_{u \in C_i, v \in C_j} (d(u, v) - x)^2 \\ = \; & |C_i||C_j| x^2 - 2x \sum_{u \in C_i, v \in C_j} d(u, v) + \sum_{u \in C_i, v \in C_j} d^2(u, v) \\ = \; & |C_i||C_j| \left(x - \frac{1}{|C_i||C_j|} \sum_{u \in C_i, v \in C_j} d(u, v) \right)^2 \\ & - \frac{1}{|C_i||C_j|} \left(\sum_{u \in C_i, v \in C_j} d(u, v) \right)^2 + \sum_{u \in C_i, v \in C_j} d^2(u, v) \end{aligned}$$

The above expression is minimized when $x = \frac{1}{|C_i||C_j|} \sum_{u \in C_i, v \in C_j} d(u, v)$. ∎

The first module of cluster center selection is an interesting problem by itself. Since the definition of cluster-to-cluster distances in the third module leads to good properties for sparsified distances, one natural clustering objective is to ensure the distances within each cluster and between any pair of clusters do not have a large spread. Specifically, for each C_i let $\max_i$ and $\min_i$ be the maximum and minimum distances between two nodes in C_i. That is, $\max_i = \max_{u,v \in C_i} d(u, v)$ and $\min_i = \min_{u,v \in C_i} d(u, v)$. Similarly, define $\max_{ij}$ and $\min_{ij}$ be the maximum and minimum distances between nodes where one is in C_i and the other is in C_j, $i \neq j$. That is, $\max_{ij} = \max_{u \in C_i, v \in C_j} d(u, v)$ and $\min_{ij} = \min_{u \in C_i, v \in C_j} d(u, v)$.

Theorem IV.2. *For $\rho > 1$, it is NP-complete to decide whether there exists a partition of V into k clusters so that the following conditions hold.*

A_i: *For all i, $1 \leq i \leq k$, $\max_i / \min_i \leq \rho$.*
$A_{i,j}$: *For all $i \neq j$, $1 \leq i, j \leq k$, $\max_{ij} / \min_{ij} \leq \rho$.*

Note that Conditions A_i constrains the intra-cluster distance variation, whereas A_{ij} constrains the inter-cluster variation. The proof of NP-completeness is via a reduction from k-coloring and details can be found in the Appendix. Since achieving a good clustering with bounded ratio of max and min distances is NP-complete and even approximating such a good clustering is not obvious, we proceed to use another related problem called the *k-center* problem.

The k-center problem takes as an input a set V of n points with specified pairwise distances, and produces a subset $L \subseteq V$ of k centers that minimize the maximum distance from a node in V to a center in L. More precisely, the aim is to minimize the maximum radius

$$\max_{v \in V} \min_{\ell \in L} d(v, \ell).$$

It is well known that k-center is NP-hard even when the distances form a metric space [12]. However, a simple greedy algorithm can guarantee an approximation ratio of 2 [13], i.e. the resulting radius is at most two times the optimal radius. and no approximation algorithm can guarantee a factor $2 - \epsilon$ for any $\epsilon > 0$ [14]. We use this simple 2-approximation as a subroutine for our MATRIXSPAR heuristic.

We use the simple iterative 2-approximation algorithm for the k-center problem [13] to identify a subset $L \subseteq V$ of k cluster centers. The algorithm works as follows. The first iteration chooses an arbitrary node in V as a center and adds to L. Each subsequent iteration chooses a node v for which the distance to the existing center set L, $\min_{\ell \in L} d(v, \ell)$, is maximized and adds v to L. Roughly speaking, this process chooses the centers as spread out as possible.

Suppose $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$ is the set of resulting centers, then $\max_{v \in V} \min_{\ell \in L} d(v, \ell)$ is within a factor 2 of the best possible [13].

Note that even an optimal solution to k-center solves a different problem from MATRIXSPAR. K-center aims to minimize the maximum radius of the clusters whereas MATRIXSPAR aims to cluster nodes with similar distances among themselves. Nevertheless, k-center is a reasonable implementation though with a somewhat different objective.

B. Distance Matrix Sparsification with Labels

The MATRIXSPAR algorithm is a general-purpose solution that can abstract a potentially large topology without any context information. However, if there are additional policies indicating that there are relevant destinations in the network

(such as `reference` in Section II-B), the algorithm can be adapted to the use of labeled data (*labeled sparsification*). In the following, this variant is named LABELEDMATRIXSPAR. In this labeled variant, the input node set V has a subset $V_R \subseteq V$ of labeled nodes (called "references").

LABELEDMATRIXSPAR can be easily solved with the MATRIXSPAR algorithm outlined above if we view the set of reference points V_R in the input as the set of k centers L in Step 1. LABELEDMATRIXSPAR then basically determines a Voronoi diagram of the labeled nodes and assigns each non-labeled node to the cluster of the closest labeled node. Labels thus significantly simplify the problem.

C. Graph Sparsification

A fundamentally different approach is GRAPHSPAR. In this case, the sparsification step operates on a graph and only calculates the final export format after sparsification has completed. For the GRAPHSPAR problem, we thus have an edge-weighted graph $G = (V, E)$ as the input where $d(u, v)$ for $(u, v) \in E$ indicates the distance between u and v. The output is an edge-weighted sparsified graph $G^s = (V^s, E^s)$ for which $|V^s| = k$ and $|E^s| \leq |E$.

Again, there are several different solutions for graph sparsification. In an earlier work [7] different heuristic were studied that used a sequence of edge contraction operations from G. In each iteration, the algorithm applies a transform rule to recalculate edge lengths of affected edges as long as there are edges below a threshold l. The quality of sparsification is measured in terms of how close the approximate distances are to the actual distances. The algorithm in [7] therefore adjusts the lengths so that a local MSE is minimized. The complete description of the local optimization algorithm can be found in [7].

D. Graph Sparsification with Labels

One of the challenges of GRAPHSPAR is that is more difficult to deal with labeled data and corresponding policies. In the paper [7] it was shown that an iterative graph sparsification algorithm can be enhanced by contraction policies, which e. g. ensure that selected nodes are not merged. Such simple policies can also ensure disjointness of references in labeled data. However, it is more difficult to enforce a more complex set of policies as introduced in Section II-B. Therefore this option is not further considered in the following.

V. EVALUATION

A. Evaluation Scenarios

In order to quantify the error resulting from sparsification, we evaluate a number of different metrics. Our comparison of different sparsification algorithms uses several topologies from different data sources. As listed in Table II, we use the artificial "Cost266" topology [15] as an example for a small network. "Cogent" from the data source [16] represents a real IP topology of a larger network, and the RocketFuel topology "7018" [17] is a well-known example for a real, large IP network. The edge length in all three topologies is calculated based on the (geographical) distance, due to the lack of other data. The original data in "Cogent" and "7018" has been cleansed in order to remove e. g. isolated notes.

We focus on sparsification without labels in Section V-B and with labels in Section V-C. For the latter, a set of selected nodes $V_R \subset V$ are labeled as references. In "Cost266", $|V_R| = 5$ nodes "Frankfurt", "London", "Madrid", "Paris", and "Stockholm" are labeled. In topology "7018", $|V_R| = 7$ nodes with the largest number of neighbors "Atlanta", "Chicago", "Dallas", "Los Angeles", "New York", "San Francisco", and "Seattle" are labeled. For "Cogent", we use the union of the above $|V_R| = 12$ reference nodes.

B. General Sparsification Error

We use the MSE defined in (1) as the primary quality measure of sparsification. In the following numerical evaluation, we verify that MATRIXSPAR outperforms GRAPHSPAR in terms of MSE. For MATRIXSPAR, we use the 3-step heuristic from Section IV-A and for GRAPHSPAR we use the local optimization method as documented in [7].

Figure 4 depicts the MSE as a function of number of clusters in the sparsified map under the three topologies introduced in Tab. II. As expected, the smaller the map and thus the number of clusters $|V^s| \leq |V|$, the larger the deviation between the original and the sparsified topology and thus the MSE. MSE for MATRIXSPAR is noticeably smaller than that for GRAPHSPAR for all three instances and all desired cluster numbers.

For the largest topology "7018", the MSE remains at a very small value even for a significant reduction. This topology has multiple instances for which several nodes are at the same geographical location and the induced edges have small default minimum distance. In most cases, the error resulting from mapping all of them into a single cluster is small. Topology "7018" includes 110 separate locations. Figure 4 confirms that the MSE significantly increases if the number of clusters is smaller than that.

Other than MSE, which is the average of $\delta(u, v)$ the distance error squared over all node pairs, we also consider the distribution of $\delta(u, v)$. Recall $\delta(u, v)$ defined to be $(d(u, v) - d^s(u, v))^2$ in Eq. (2) is the squared difference between the actual and sparsified distances between node pair u and v. Fig. 5 shows the cumulative distribution function (CDF) of $\delta(u, v)$. More specifically, each point (x, y) on the CDF curve indicates a fraction y of the node pairs have square errors up to x. For the cumulative distribution, the higher the curve, the smaller the error. Fig. 5 is for 20 resulting clusters of the "Cogent" instance, and shows the advantage of MATRIXSPAR over GRAPHSPAR. For a range of values of $|V^s|$ for the Cogent, Cost266 and 7018 topologies we observe CDF curves that are qualitatively similar to those in Fig. 5. We therefore do not repeat them here.

TABLE II. LIST OF TOPOLOGIES IN USED IN THE EVALUATION

Topology	Type	Vertices $\|V\|$	Edges $\|E\|$	Cost	Source
Cost266	Artificial	37	144	Distance	[15]
Cogent	IP	180	420	Geogr. dist.	[16]
7018	IP	627	4102	Geogr. dist.	[17]

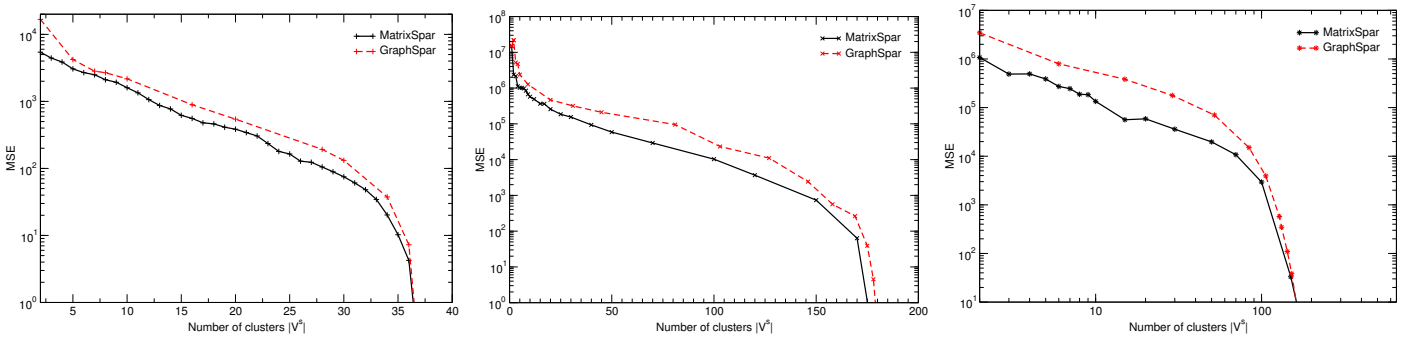

Fig. 4. Comparison of the MSE, MATRIXSPAR vs GRAPHSPAR. (Left) Cost266, (Middle) Cogent, (Right) 7018.

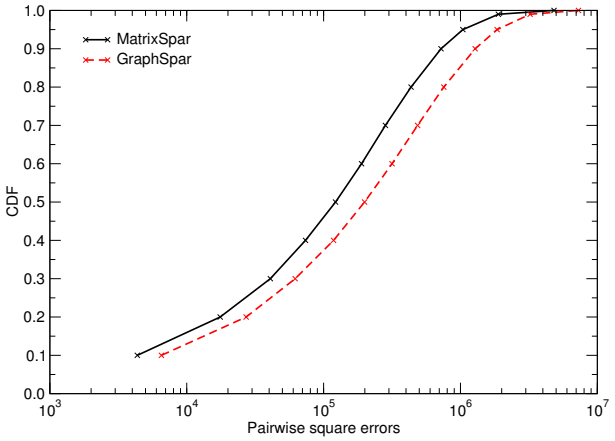

Fig. 5. CDF of the pairwise square errors for "Cogent" with $|V^s| = 20$

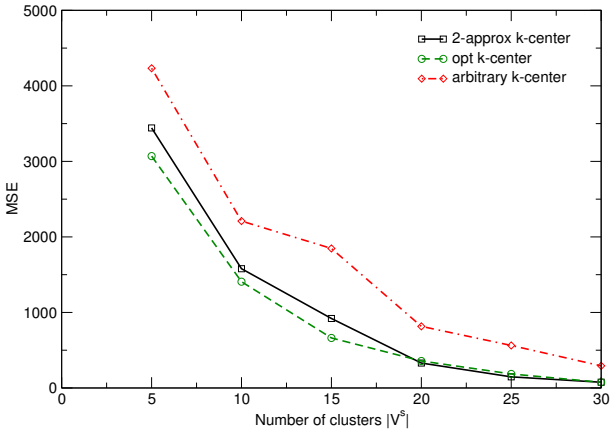

Fig. 6. Comparison of MSE under different clustering methods for MATRIXSPAR, Cost266.

In Figure 6, we further discuss the implementation of the first step in the 3-step heuristic for MATRIXSPAR. Recall the first step is identifying k cluster centers, for which we use a k-center problem solution. We have discussed that in fact the k-center problem has a related but different objective from MATRIXSPAR. Here we use experimental results to illustrate that whether we use the simple 2-approximation to find k centers (as described in Section IV-A) or use the optimal k-center solution, the resulting MSE values are very similar. For small instances such as "Cost266", the k-center problem can be solved optimally using an integer linear program (ILP) formulation via a commercial ILP solver such as CPLEX [18]. On the other hand, as expected, it is not the case that MSE is completely insensitive to the choice of cluster centers. The top curve in Figure 6 is the resulting MSE of one arbitrary choice of centers, which are significantly worse than the choices by k-center, approximation or exact.

C. Sparsification Error for Labeled Data

The MSE as defined in Eq. (1) characterizes the deviation of the sparsified network topology compared to the original as a whole. In addition, we also consider error metrics adapted to the use of a topology manager for application guidance. In this case, an application (e. g., a CDN) is typically only interested in the distances to a subset of nodes $V_R \subset V$ that are labeled as reference node. There destinations can be provided by the policies introduced in Section II-B (i. e., `reference`).

Let $V_N = V \setminus V_R$ be the remaining nodes. For any $u \in V_N$, let $r(u) = argmin_{v \in V_R} d(u, v)$ be the reference node closest to u. In case of a CDN as an example user of a topology manager, $r(u)$ has an intuitive meaning: If V_R represents the locations of the references, $r(u)$ provides for all other network locations the optimal cache regarding the distance metric used in the topology. The minimum distance to a reference node is then given by $d(u, r(u))$. Let $r^s(u) = argmin_{v \in V_R} d^s(u, v)$ be the reference node closest to u after sparsification where ties are broken arbitrarily. Note that $r(u)$ may not be the same as $r^s(u)$. If the reference nodes are known and if the LABELEDMATRIXSPAR method is used, each cluster contains exactly one reference node. The sparsified distance is therefore $d^s(u, r^s(u))$. Distance sparsification affects the distance to the closest reference node. This error can be measured by the *Mean Square Error to Reference* (MSER) metric.

$$MSER = \frac{1}{|V_N|} \sum_{u \in V_N} (d(u, r(u)) - d^s(u, r^s(u)))^2. \quad (4)$$

In Fig. 7, we present numerical results for the MSER given by Eq. (4), using Cost266 as example. For varying number of clusters $|V^s|$, we run MATRIXSPAR which is label unaware, but compute MSER for every possible choice of five reference nodes and the curve in Fig. 7 plots the average. As to be expected, MSER has a similar behavior like MSE, i. e., the more clusters, the smaller the error. The results for GRAPHSPAR are similar and therefore omitted in this diagram.

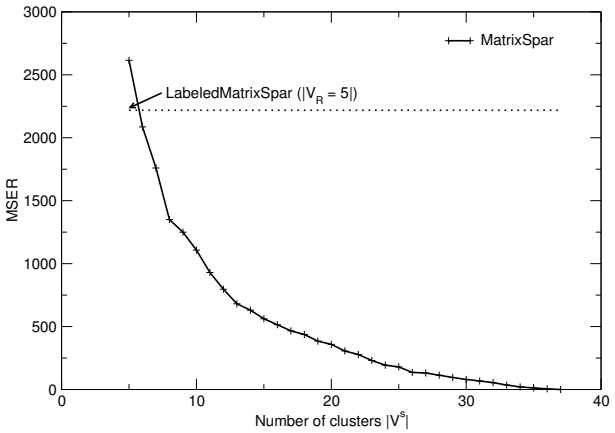

Fig. 7. Mean Square Error to Reference (MSER) for the Cost266 topology with $|V_R| = 5$ reference nodes.

Of interest is the case in which $|V^s| = 5$. Recall we have a set of $|V_R| = 5$ reference nodes for Cost266 specified at the beginning of this section, and we refer to them as $V_R^{Cost266}$. We compare the MSER produced by LABELEDMATRIXSPAR taking the $V_R^{Cost266}$ as input, against the average MSER produced by MATRIXSPAR for $|V^s| = 5$. Not surprisingly, the former outperforms the latter since the former is label aware. This can be seen in Figure 7 for $|V^s| = 5$.

Fig. 7 also shows that MATRIXSPAR although unaware of the reference nodes can provide a smaller average MSER if the number of clusters is allowed to be larger than the number of reference nodes. This is further confirmed in Table III, which provides results for all three test topologies with the reference nodes introduced in Section V-A. Our results show that, in general, using MATRIXSPAR with a number of clusters twice the number of reference nodes ($|V^s| = 2|V_R|$) results in a much smaller error than label-aware Voronoi clustering. There results obviously depend on the position of reference nodes.

While knowing reference nodes helps to reduce the error in distance sparsification, in practice such knowledge may come at a cost. Obtaining the reference node set and subsequent policy setup may require an additional information exchange between the user of the topology manager and its operator, thus increasing the overall system complexity. Our results thus show that a topology exposure system unaware of the actual destinations can still provide reasonably accurate guidance with low error as long as a moderate number of clusters is used, in particular if it is sufficiently larger than the number of important destinations in the network.

D. Further Improvements and Open Issues

There are various alternatives to the solutions presented in this initial study. For instance, instead of k-center clustering in MATRIXSPAR, one could also apply other known graph clustering heuristics. Yet, there may be only limited benefit from designing advanced algorithms for the label-unaware case, since the system cannot determine optimality in this case. The label-aware case is easier to solve, but further improvement is possible in this case, too. For instance, in this work we only consider static labels. Changes of labels (e.g., resulting of the deployment of additional CDN caches) will require a dynamic, online algorithm.

TABLE III. EXAMPLE MSER RESULTS FOR THE THREE TEST CASES

Topology	LABELEDMATRIXSPAR $\lvert V^s\rvert = \lvert V_R\rvert$	MATRIXSPAR $\lvert V^s\rvert = 2\lvert V_R\rvert$
Cost266	3552	1601
Cogent	859009	186960
7018	126645	64315

Future experimentaion is needed to proof that our sparsification algorithms indeed operate well with real applications. This paper focuses on additive distance or delay metrics as a starting point to understand the algorithmic issues. As alternative, the congestion status on links would also be an interesting but very volatile metric. In addition, the evaluation in this paper does not consider complex routing setups, e.g., the combination of BGP and Interior Gateway Protocols (IGP).

Concerning the architecture, we focus on the exposure of abstract topology information to applications that only query data. Sparsification methods could also be used in an SDN controller or a PCE, but we do not consider the implications of control action based on sparsified topology, such as congestion-aware routing. There are also options for more complex policies, but this is beyond the scope of this work.

VI. OTHER RELATED WORK

Shortest paths calculation, approximate distance preservation, embedding, clustering, and routing are closely related topics studied by diverse communities. A thorough literature review of such areas would be overwhelming, but we give a few examples to illustrate related work.

Graph sparsification is well studied in the literature. From the theoretical front, different notions of sparsification have been considered [19] including distance sparsifier for preserving pair-wise distance, e.g. [20], cut sparsifier for preserving cut values, e. g. [21], and spectral sparsifier for preserving graph Laplacian, e. g. [22]. Distance preservation is closest to our interest. However, most previous work focuses on edge sparsification, which means the node set is untouched, but the graph becomes sparser with fewer edges. A classic example is the t-spanner problem, which is to find a smallest subset of the edges such that the distance is stretched by a factor of t at most. Formally, *stretch* is the *worst* ratio between the approximate distance to the actual distance $\max_{u,v} \frac{d^s(u,v)}{d(u,v)}$. A tradeoff between t and the number of edges in the sparsified graph, and the time complexity are studied in a sequence of papers, e. g. [20], [23]. We note that worst multiplicative ratio may not fit well in use cases where the absolute value of network distance matters to the data consumer (e. g., geographic distance, delay, or routing weights). In addition, the worst distortion of one distance may not be representative of the distortion of all distances.

Therefore we have considered additive measures such as the sum square difference in our study. Relatively few papers focus on vertex sparsification of a graph. Please see [24] for a summary. Most of the related work on sparsification takes a graph as input and outputs a sparsified graph. This is different to our solution that calculates a sparsified distance matrix. Furthermore, to the best of our knowledge distance preservation is not studied in the context of vertex sparsification.

Somewhat related goals are seen in *compact routing* where the idea is to reduce the size of routing tables while minimizing the increase in the length of paths between nodes. A good overview of the compact routing literature can be found in [25]. While compact routing affects the actual routing of packets, the problem we consider does not change the routing but affects such things as the determination of nearest neighbors or other computations that rely on path lengths.

If the abstraction policies provide labels with additional context information regarding relevent destinations, the principle of Voronoi diagrams can be applied [26]. The authors are not aware of related published work on details of a policy design for a topology abstraction system, apart from [7].

VII. Conclusion

This paper proposes algorithms and a policy system for abstracting network information. Both are important components of a topology exposure system that improves application-level resource selection. Our proposed solution for privacy enforcement extends and generalizes the concept of route import filters. A further size reduction of the exposed data can be achieved by distance sparsification algorithms. We propose heuristics for matrix and graph sparsification. We believe that matrix sparsification is new and significantly different from other sparsification literature. As sparsification method we propose a modular three-step algorithmic framework. For the first two steps we use intuitive algorithms, whereas our proposed algorithm for the third step is optimal given the output of the previous steps. In order to analyze the impact of abstraction we quantify the sparsification error. We observe the advantage of matrix sparsification over the graph counterpart, and the trade-off between accuracy and the size of sparsified representation. We also show that even without label information in many cases one can significantly reduce the size of a topology without dramatically impacting distance accuracy.

Appendix

NP-completeness proof of Theorem IV.2.

Proof: Clearly the problem is in NP since checking that a given partition satisfies the A_i and $A_{i,j}$ constraints can easily be done in polynomial time.

To show that it is NP-hard we describe a reduction from k-COLORING. Let $G = (V, E)$ be an instance of k-COLORING. From G we define an instance $I = (V, d, k, \rho)$ of BOUNDED RATIO CLUSTERING.

For each $v \in V$ define nodes v and v' in V where $d(v, v') = \epsilon$ and $0 < \epsilon < 1$. Let $\alpha = \min\{2, \rho\} > 1$. For each $e = uv \in E$ we define $d(u, v) = d(u, v') = d(u', v) = d(u', v') = M$ where $M = \epsilon\alpha\rho$. Therefore if $uv \in E$ then in any valid clustering u and u' must be in different clusters than v and v' since otherwise if say u and v were in cluster C_i then $\max_i / \min_i = M/\epsilon = \alpha\rho > \rho$ since $\alpha > 1$ and hence violates an A_i conditions. Let $M' = \epsilon\rho$ and for $uv \notin E$ define $d(u, v) = d(u, v') = d(u', v) = d(u', v') = M'$.

Note that for any edge $e = uv$ whose length is ϵ or M', it is straightforward to check that any path between u and v has total length at least that of e. Suppose $e = uv$ has distance M. Then any path between u and v (other than e) contains at least two edges that both have length M' and so the total length of any such path is at least $2M' = 2\epsilon\rho \geq \alpha\epsilon\rho = M$. Thus the distance measure d satisfies the triangle inequality.

Suppose there is a k-coloring C of G. Then put u, u' in C_i if u is given color i in C. We wish to show that this defines a valid clustering.

For any two nodes in C_i the distance between them is ϵ or M'. But

$$M'/\epsilon = \epsilon\rho/\epsilon = \rho$$

so the nodes within each cluster do not violate the A_i conditions.

If two nodes u and v are in different clusters, then $d(u, v) = M'$ if $uv \notin E$ or $d(u, v) = M$ if there is an edge $uv \in E$. Then the ratio of maximum to minimum distance between such points is

$$M/M' = \epsilon\rho\alpha/\epsilon\rho = \alpha \leq \rho.$$

Therefore the intercluster distances do not violate the $A_{i,j}$ conditions. That is, we have k clusters forming a valid clustering.

Suppose on the other hand that we have a valid clustering S of V of size k. We color node u with color i if $u \in C_i$ in S. We wish to show that for any edge $uv \in E$, u and v have different colors (or equivalently, if $u \in C_i$ and $v \in C_j$ according to S, then $i \neq j$). But $d(u, v) = M$ and $d(u, u') = \epsilon$ and so $\max_{ij} / \min_{ij} \geq M/\epsilon = \rho\alpha > \rho$ contradicting the assumption that S was a valid clustering. ■

Acknowledgment

The authors wish to thank Iraj Saniee and Thomas Voith for many helpful discussions, and Volker Hilt and Markus Hofmann for their support.

References

[1] A. Atlas, J. Halpern, S. Hares, D. Ward, and T. Nadeau, "An Architecture for the Interface to the Routing System," Internet Draft, work in progress, 2014.

[2] D. King and A. Farrel, "A PCE-based Architecture for Application-based Network Operations," Internet Draft, work in progress, 2014.

[3] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, and R. Wound, "Application-Layer Traffic Optimization (ALTO) Protocol," RFC 7285, 2014.

[4] M. Stiemerling, S. Kiesel, S. Previdi, and M. Scharf, "ALTO Deployment Considerations," Internet Draft, work in progress, 2014.

[5] R. Ravindran, C. Huang, and K. Thulasiraman, "Topology Abstraction Service for IP-VPNs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 184–197, 2013.

[6] M. Scharf, V. Gurbani, T. Voith, M. Stein, W. Roome, G. Soprovich, and V. Hilt, "Dynamic VPN optimization by ALTO guidance," Proc. EWSDN workshop, 2013.

[7] M. Scharf, T. Voith, M. Stein, and V. Hilt, "ATLAS: Accurate Topology Level-of-Detail Abstraction System," in *IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–5.

[8] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," RFC 6020, 2010.

[9] H. Gredler, J. Medved, S. Previdi, A. Farrel, and S. Ray, "North-Bound Distribution of Link-State and TE Information using BGP," Internet Draft, work in progress, 2014.

[10] M. Caesar and J. Rexford, "BGP Routing Policies in ISP Networks," *IEEE Network*, vol. 19, no. 6, pp. 5–11, 2005.

[11] *7750 SR OS Routing Protocols Guide*, Alcatel-Lucent, 2014.

[12] V. Vazirani, *Approximation Algorithms*. Berlin: Springer, 2003.

[13] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science, Elsevier Science B.V.*, vol. 38, pp. 293–306, 1985.

[14] D. S. Hochbaum and D. B. Shmoys, "A unified approach to approximation algorithms for bottleneck problems," *Journal of the ACM (JACM)*, vol. 33, no. 3, pp. 533–550, 1986.

[15] "Networks and Problem instances of SNDlib," http://sndlib.zib.de, 2006.

[16] "The Internet Topology Zoo," http://www.topology-zoo.org, 2013.

[17] "Rocketfuel: An ISP Topology Mapping Engine," http://research.cs.washington.edu/networking/rocketfuel/, 2002.

[18] "ILOG CPLEX C++ API 11.0 reference manual," http://www-eio.upc.edu/lceio/manuals/cplex-11/pdf/refcppcplex.pdf, 2007.

[19] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi, "Graph sparsification by edge-connectivity and random spanning trees," in *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, 2011.

[20] E. Cohen, "Fast Algorithms for Constructing t-Spanners and Paths with Stretch t," *SIAM Journal on Computing*, vol. 28, no. 1, pp. 210–236, 1999.

[21] A. A. Benczur and D. R. Karger, "Approximate s-t min-cuts in $\tilde{O}(n^2)$ time," in *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*.

[22] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, 2004.

[23] I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discrete and Computational Geometry*, vol. 9, no. 1, pp. 81–100, 1993.

[24] A. Moitra, "Vertex sparsification and universal rounding algorithms," Ph.D. dissertation, MIT, 2011.

[25] M. Enachescu, M. Wang, and A. Goel, "Reducing maximum stretch in compact routing," in *INFOCOM*, 2008, pp. 336–340.

[26] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.