Active Learning of Multiple Source Multiple Destination Topologies

Pegah Sattari, *Member, IEEE,* Maciej Kurant, Animashree Anandkumar, *Member, IEEE,*
Athina Markopoulou, *Senior Member, IEEE,* and Michael Rabbat, *Member, IEEE*

Abstract—We consider the problem of inferring the topology of a network with M sources and N receivers (hereafter referred to as an M-by-N network), by sending probes between the sources and receivers. Prior work has shown that this problem can be decomposed into two parts: first, infer smaller subnetwork components (*i.e.,* 1-by-N's or 2-by-2's) and then merge these components to identify the M-by-N topology. In this paper, we focus on the second part, which had previously received less attention in the literature. In particular, we assume that a 1-by-N topology is given and that all 2-by-2 components can be queried and learned using end-to-end probes. The problem is *which* 2-by-2's to query and *how* to merge them with the given 1-by-N, so as to exactly identify the 2-by-N topology, and optimize a number of performance metrics, including the number of queries (which directly translates into measurement bandwidth), time complexity, and memory usage. We provide a lower bound, $\lceil \frac{N}{2} \rceil$, on the number of 2-by-2's required by any active learning algorithm and propose two greedy algorithms. The first algorithm follows the framework of multiple hypothesis testing, in particular Generalized Binary Search (GBS), since our problem is one of active learning, from 2-by-2 queries. The second algorithm is called the Receiver Elimination Algorithm (REA) and follows a bottom-up approach: at every step, it selects two receivers, queries the corresponding 2-by-2, and merges it with the given 1-by-N; it requires exactly $N-1$ steps, which is much less than all $\binom{N}{2}$ possible 2-by-2's. Simulation results over synthetic and realistic topologies demonstrate that both algorithms correctly identify the 2-by-N topology and are near-optimal, but REA is more efficient in practice.

Index Terms—Adaptive Sensing Algorithms, Inference and Estimation on Graphs, Applications of Statistical Signal Processing Techniques, Sequential Learning, Active Hypothesis Testing, Network Monitoring, Internet, Tomography.

I. INTRODUCTION

KNOWLEDGE of network topology is important for network management, diagnosis, operation, security, and

performance optimization [1–6]. In this paper, we consider a tomographic approach to topology inference, which assumes no cooperation from intermediate nodes and relies on end-to-end probes to infer internal network characteristics, including topology [4]. Typically, multicast or unicast probes are sent and received between sets of sources and receivers at the edge of the network, and the topology is inferred based on the number and order of received probes, or more generally, using some metric or correlation structure. An important performance metric is measurement bandwidth overhead: it is desirable to accurately infer the topology using a small number of probes.

In this paper, we focus on the problem of multiple-source multiple-destination topology inference: our goal is to infer the internal network (M-by-N) topology by sending probes between M sources and N receivers at the edge of the network. Prior work [1–3] has shown that this problem can be decomposed into two parts: first, infer smaller subnetwork components (*e.g.,* multiple 1-by-N's or 2-by-2's) and then merge them to identify the entire M-by-N topology.

Significant progress has been made over the past years on the decomposition as well as the first part of the problem, *i.e.,* inferring smaller components (1-by-N's or 2-by-2's) using active probes. One body of work developed techniques for inferring 1-by-N (*i.e.,* single-source tree) topologies using end-to-end measurements [7–15]. Follow-up work [1–3] showed that an M-by-N topology can be decomposed into and reconstructed from a number of two-source, two-receiver (2-by-2) subnetwork components or "quartets". In [1, 2], a practical scheme was proposed to distinguish between some quartet topologies using back-to-back unicast probes. In our recent work [16, 17], we proposed a method to exactly identify the topology of a quartet in networks with multicast and network coding capabilities.

In this paper, we focus on the second part of the problem, namely selecting and merging smaller subnetwork components to exactly identify the M-by-N, which has received significantly less attention than the first part. Existing approaches developed for merging the quartets [1, 3] have several limitations, including not being able to exactly identify the M-by-N topology and/or being inefficient (*e.g.,* requiring to send probes over all $\binom{N}{2}$ possible quartets). In this paper, we formulate the problem as active learning, characterize its complexity, and follow principled approaches to design efficient algorithms to solve it. This complexity is important from both theoretical (a fundamental property of the topology inference problem) and practical (it determines the measurement bandwidth overhead,

running time and memory usage) points of view. These costs can become particularly important when we need to infer large or dynamic topologies using active measurements, and an efficient algorithm is required for that.[1]

More specifically, we start from the problem of 2-by-N topology inference, which is an important special case and can then be used as a building block for inferring an M-by-N. Consistently with [1], we assume that a (static) 1-by-N topology is known (*e.g.,* using one of the methods in [4, 7–15, 23]). Then we query the quartet component by sending end-to-end probes between the two sources and the two receivers, and we learn its topology using some of the methods in [1, 2, 16, 17, 24–28][2]. The problem then becomes one of active learning: "*which* quartets to query and *how* to merge them with the given 1-by-N, so as to exactly identify the 2-by-N topology and optimize a number of performance metrics, including the number of queries (thus the measurement bandwidth), time complexity, and memory usage." Our contributions are as follows:

1) We provide a lower bound of $\lceil \frac{N}{2} \rceil$ on the number of quartets required by *any* active learning algorithm in order to identify the 2-by-N. This characterizes the inherent complexity of the problem and also serves as a rough baseline for assessing the performance of practical algorithms.

2) We formulate the problem within the framework of multiple hypothesis testing and develop an active learning algorithm based on Generalized Binary Search (GBS). This is the natural framework to pose the problem; however, we evaluate the performance of this algorithm via simulation and show that the computational complexity is high in practice.

3) As an alternative, we design an efficient Receiver Elimination Algorithm (REA), which follows a greedy bottom-up approach and provably identifies the 2-by-N topology by querying exactly $N - 1$ quartets. From the active probing perspective, this is attractive since only $N - 1$ queries are required, which is much lower than all $\binom{N}{2}$ possible quartets one could query. This directly results in very low measurement bandwidth, which is the main performance metric in active monitoring.

We compare the two algorithms to each other and to the lower bound via simulation over both synthetic and realistic topologies. The results show that both algorithms can exactly identify the topology and are near-optimal in terms of active measurement bandwidth. Between the two, the Receiver Elimination Algorithm is found to be very efficient in terms of running time and memory usage, and is, therefore, recommended for practical implementation.

The rest of the paper is organized as follows. Section II summarizes related work. Section III provides the problem statement and terminology. Section IV provides a lower bound on the number of quartets required by any algorithm. Section V proposes a greedy algorithm based on the GBS framework and evaluates its performance via simulation. Section VI proposes the greedy REA, analyzes its correctness and performance, and compares it to GBS in simulation. Section VII discusses possible extensions. Section VIII concludes the paper.

II. RELATED WORK

There is a large body of prior work on inference of network topology. The most closely related to this paper are the ones using active measurements and network tomography.

Tomographic approaches. A survey of *network tomography* can be found in [4]. Tomographic approaches rely only on end-to-end measurements to infer internal network characteristics, which may include link-level parameters (such as loss and delay metrics) or the network topology [29–34]. In this paper, we focus on inferring the network topology. Most tomographic approaches rely on probes sent from a single source in a tree topology [7–15] and feed the number, order, or a monotonic property of received probes as input to statistical signal-processing techniques.

In [1–3], the authors formulated the multiple source multiple destination (M-by-N) tomography problem by sending probes between M sources and N receivers. It was shown that an M-by-N network can be decomposed into a collection of 2-by-2 components, also referred to as quartets [5, 6]. Coordinated transmission of back-to-back unicast probes from the two sources and packet arrival order measurements at the two receivers were used to infer some information about the quartet topology. Assuming knowledge of M 1-by-N topologies and the quartets, it was also shown how to merge a second source's 1-by-N tree topology with the first one. The resulting M-by-N topology is not exact, but bounds were provided on the locations of the points where the two 1-by-N trees merge with each other. This approach also requires a large number of probes for statistical significance, similar to many other methods [7–11]. Compared to [1], our work is different in that (i) we assume perfect knowledge of the quartets, thus we identify the topology accurately; (ii) we focus on the efficiency of active learning, *i.e.,* selecting and merging the quartets, which has not been studied before. To the best of our knowledge, the only other merging algorithm proposed in the literature is [1, 3]. However, the merging was not efficient since all possible quartets were queried exhaustively.

In our prior work [16, 17], we revisited the problem of topology inference using end-to-end probes in networks where intermediate nodes are equipped with multicast and network coding capabilities. We built on [1] and extended it, using network coding at intermediate nodes to deterministically distinguish among all possible quartet topologies, which was not possible before. While in [16, 17], we focused on inferring the quartets fast and accurately, here we assume that any quartet can be queried and learned, and focus on efficiently selecting and merging the quartets to infer the larger topology. To the best of our knowledge, this work is the first to look at this aspect of the problem.

---

[1]Examples of networks where up-to-date topology information, and thus dynamic mapping of the topology, is required include the following: detection systems that detect Internet faults [18, 19] or prefix hijacks [20] and require frequent measurements of Internet paths; content distribution networks that need to continuously monitor the topology in order to select the *best* content server for user requests [21]; and overlay networks that need to monitor the topology to select the best overlay routing [22].

[2]Other techniques may also be developed in the future as this is still an active research area. However, this is out of the scope of this paper (see Section III).

There also exists a rich body of work on *multiple hypothesis testing* for active learning problems where queries are selected adaptively. One of the contributions of this paper is to formulate this problem in that framework and design an algorithm based on one such active learning scheme, GBS [35–37], which we describe in detail in Section V.

Topology inference problems have also been studied in the context of *phylogenetic* trees [38, 39]. The work in [6] built on [39] and proposed robust algorithms for multiple source tree topology inference. The work in [5] inferred the topology of sparse random graphs using end-to-end measurements between a small subset of nodes. However, the quartet structures and the way we measure them are different in our case due to the nature of active probing in network tomography (see problem formulation in Section III).

Traceroute-based approaches. An alternative to tomographic approaches is `traceroute`-based techniques, which rely on cooperation of nodes in the middle of the network, in order to connect the ids of nodes along paths and reconstruct the topology [25–28]. These approaches face their own set of challenges: not all intermediate nodes cooperate by responding, many of them have multiple network interfaces (ids), and `traceroute` is often turned off for security reasons. Therefore, `traceroute`-based methods must deal with missing or incomplete data and alias problems. Regardless, the point of this paper is not to compare the tomographic approaches against the `traceroute`-based approaches, but to provide active learning algorithms that can probe/query the network in an efficient way. Querying the network can be achieved via end-to-end probes, `traceroute`, or even in a passive way. As long as we can query 2-by-2 components, the active learning approach should be applicable and useful in minimizing the cost of all such approaches.

Relation to the conference version. This journal paper builds on our conference paper in [40]. In addition to revisions and elaborating on parts of the writing, new materials/contributions in this paper include the following: the formulation of the problem in the GBS framework as well as the performance evaluation of the GBS algorithm via simulation, and its comparison against REA.

III. PROBLEM STATEMENT

M-by-N Topology to be inferred. Consider an M-by-N topology as a directed acyclic graph (DAG), between M source nodes $\mathcal{S} = \{S_1, ..., S_M\}$ and N receiver nodes $\mathcal{R} = \{R_1, ..., R_N\}$. We denote this M-by-N topology by $G_{\mathcal{S}\times\mathcal{R}}$. Note that $G_{S_i\times\mathcal{R}}$, $i = 1, ..., M$, is a 1-by-N tree. Similar to [1–3], we assume that a predetermined routing policy maps each source-destination pair to a unique route from the source to the destination. This implies the following three properties, first stated in [1]:[3]

A1 For every source S_i and every receiver R_j, there is a unique path P_{ij}.

---

[3]These assumptions are realistic, the same as in [1–3], and consistent with the destination-based routing used in the Internet: each router decides the next hop taken by a packet using a routing table lookup on the destination address. We further assume that the network does not employ load balancing.
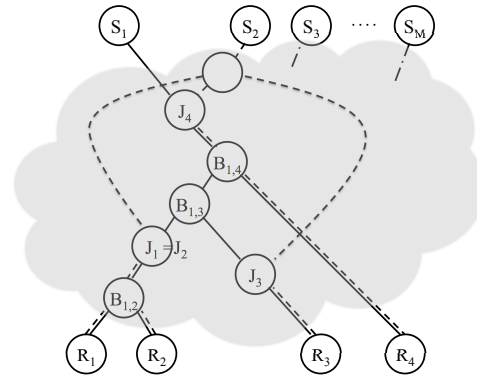

Fig. 1. An example 2-by-4 topology. The solid lines and branching points $B_{i,j}$'s depict the S_1 tree topology, $G_{S_1\times\mathcal{R}}$. J_i is a joining point, where P_{2i} (indicated by the dashed lines) joins $G_{S_1\times\mathcal{R}}$. An example quartet is the part of the network connecting S_1, S_2 to R_1, R_2, which is type 1 since both J_1 and J_2 lie above the branching point of R_1 and R_2 in $G_{S_1\times\mathcal{R}}$, *i.e.*, $B_{1,2}$.

A2 Two paths P_{ij} and P_{ik}, $j \neq k$, branch at a *branching point B*, and they never merge again.

A3 Two paths P_{ik} and P_{jk}, $i \neq j$, merge at a *joining point J*, and they never split again.

We are interested in inferring the logical topology[4], defined by the branching and joining points defined above. We present most of our discussion in terms of $M = 2$, *i.e.,* inferring a 2-by-N topology $G_{\mathcal{S}\times\mathcal{R}}$, $\mathcal{S} = \{S_1, S_2\}$; an M-by-N topology, $\mathcal{S} = \{S_1, ..., S_M\}$, can then be constructed by merging smaller structures, as we describe in Section VII.

Example 1: Fig. 1 illustrates an example 2-by-N topology with $N = 4$. The logical tree topology of S_1 is shown by solid lines and branching points $B_{i,j}$'s. Each J_i depicts a joining point, where the path from S_2 to receiver R_i (indicated by the dashed lines) joins the S_1 tree. For example, the path from S_2 to R_1 joins the S_1 tree at a point between $B_{1,3}$ and $B_{1,2}$, whereas the path to R_4 joins at a point above $B_{1,4}$. ∎

Quartet Components. In [1], it has been shown that an M-by-N topology can be decomposed into a collection of 2-by-2 subnetwork components, which, in this paper, we call *quartets*, following the terminology in [5, 6]. Each quartet can be of four possible types, as shown in Fig. 2. We refer to Fig. 2 (a), (b), (c), and (d) as types 1, 2, 3, and 4, respectively. Note that in type 1, the joining points for both receivers coincide ($J_1 \equiv J_2$) and the branching points for both sources coincide ($B_{1,2}^1 \equiv B_{1,2}^2$). However, the other three types (2, 3, and 4), have two distinct joining points and two distinct branching points.

In order to infer the type of a quartet between two sources S_1, S_2 and two receivers R_i, R_j, a set of probes must be sent from S_1, S_2 to R_i, R_j. The received probes can then be processed using techniques such as the ones developed in: [1, 2] (which distinguish type 1 from types 2, 3, 4 by sending back-to-back unicast probes); [16, 17] (which distinguish among all four types exploiting multicast and network coding); [24] (which can exactly infer the topology of a super-source to

---

[4]A logical topology is obtained from a physical topology by ignoring nodes with in-degree = out-degree = 1. Such nodes cannot be identified and network tomography always focuses on inferring logical topologies.
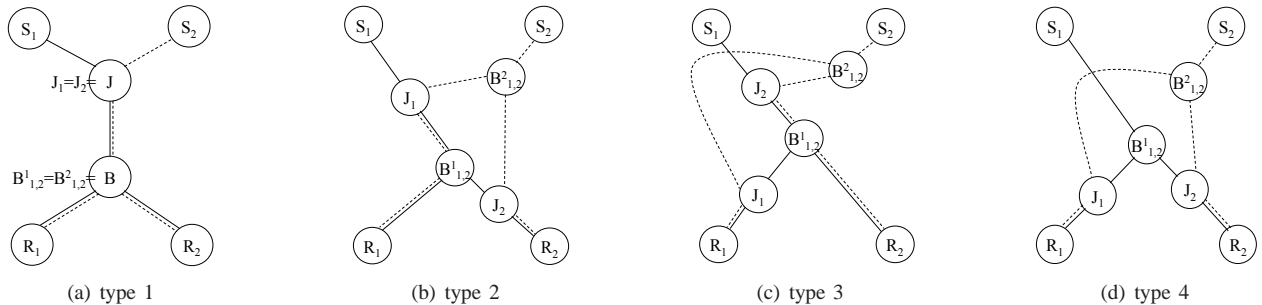
Fig. 2. The four possible types of a quartet (2-by-2 subnetwork component). There are two sources S_1 and S_2 multicasting packets x_1 and x_2, respectively, to two receivers R_1 and R_2. All links are directed downwards, but arrowheads are omitted to avoid cluttering. The 1-by-2 topology of S_1 is a tree composed of $S_1, B^1_{1,2}, R_1, R_2$. Similarly, the 1-by-2 tree rooted at S_2 is $S_2, B^2_{1,2}, R_1, R_2$. J_1 and J_2 are joining points, where the paths from S_2 to R_1 and R_2 join/merge with S_1's tree topology.

two receivers using network coding); `traceroute` [25–28] from the two sources to the two receivers; or other techniques that may be developed in the future, since this is still an active research area. We consider the design of these techniques to be out of the scope of this paper and we focus on their use by active learning algorithms to perform a *query*, *i.e.*, to learn a quartet type by sending and processing a set of active probes.

Being able to query the type of a quartet enables inference of an M-by-N topology in two steps, as follows: first infer the type of each quartet, and then merge these quartets to identify the original topology. Indeed, knowing the type of the quartet, we can use Fig. 2 to infer the relative location of joining and branching points. For example, knowing that the quartet is of type 1 implies that (i) the two joining points coincide $J_1 \equiv J_2$, (ii) the two branching points coincide $B^1_{1,2} \equiv B^2_{1,2}$, and (iii) the joining point is above the branching point. Similar inferences can be made from the other types.

Problem Statement. Consistently with [1], we assume that $G_{S_1 \times \mathcal{R}}$ (*i.e.*, the 1-by-N tree topology rooted at S_1, which contains only branching points) is known (*e.g.*, using one of the methods in [4, 7–15, 23]). We also assume that the type of the quartet between S_1, a new source S_2, and any two receivers can be queried and learned, as explained above.

Given (i) $G_{S_1 \times \mathcal{R}}$ and (ii) the ability to query the quartet type between S_1, S_2, and any two receivers R_i, R_j, our goal is to *identify* all joining points, $\mathcal{J}_N = \{J_1, J_2, ..., J_N\}$, where the paths from S_2 to each receiver join the tree describing paths from S_1 to the same set of receivers.[5] Identifying a joining point J_i (for receiver R_i) means locating J_i on a single logical link, between two branching points on $G_{S_1 \times \mathcal{R}}$. *E.g.*, in Fig. 1, the path from S_2 to R_1 joins the S_1 tree at a point between nodes $B_{1,3}$ and $B_{1,2}$; *i.e.*, J_1 is located on the link $(B_{1,3}, B_{1,2})$.

We achieve this goal via active learning: we start from the given, static, 1-by-N topology $G_{S_1 \times \mathcal{R}}$, and proceed by updating it in steps. In each step, we select which quartet to query (*i.e.*, which two receivers to send probes to, from sources

S_1 and S_2)[6], and learn its type (after sending and processing the received probes, we have essentially queried and learned the type of that quartet). We then merge this quartet with the known topology so far. We continue until identifying the entire 2-by-N. The goal is to exactly identify the 2-by-N topology while minimizing the number of queries (*i.e.*, set of probes sent to measure the quartets). This metric is important because it directly translates into measurement bandwidth. Additional performance metrics that it is desirable to keep low include: merging complexity and memory usage.

IV. LOWER BOUND

First, we provide a lower bound on the number of quartets required by any active learning algorithm to identify the 2-by-N topology. This lower bound clearly depends on the topology we want to identify and serves as a baseline for assessing the performance of the proposed algorithms.

Theorem 4.1: Given $G_{S_1 \times \mathcal{R}}$, the number of quartets required to be queried by any algorithm in order to identify all the joining points in $G_{\mathcal{S} \times \mathcal{R}}$, $\mathcal{S} = \{S_1, S_2\}$, is at least $\lceil \frac{N}{2} \rceil$.

Before proving the theorem, let us discuss some examples that illustrate the intuition and that this bound is not tight.

Example 2: Fig. 3(a) shows a 2-by-N topology with $N = 4$, which requires querying exactly $\frac{N}{2} = 2$ quartets in order to uniquely identify all the joining points. This is because, in this particular topology, knowing the types of (R_1, R_2) and (R_3, R_4) is sufficient for identifying all four joining points. Indeed, (R_1, R_2) is of type 4, which, according to Fig. 2, means that both J_1 and J_2 lie below $B_{1,2}$; also (R_3, R_4) is type 4, which means that both J_3 and J_4 are below $B_{3,4}$. Thus, each joining point is identified on a single logical link. ∎

Example 3: Fig. 3(b) shows an example where $\frac{N}{2} = 2$ quartets are not sufficient and 3 quartets are required to identify all the joining points. There exist $\binom{4}{2} = 6$ possible quartets in this topology, from which $\binom{6}{2} = 15$ pairs of quartets can be selected; one can check that none of the 15 possible pairs can uniquely identify all the joining points. For example, let us consider (R_1, R_2). Since it is of type 1, Fig. 2 indicates that $J_1 \equiv J_2$ and both of them lie above $B_{1,2}$. However, there is more than a single link above $B_{1,2}$; therefore, we continue

---

[5]Note that we do not need to identify the branching points of S_2 because the tree topology of S_2, like S_1, is given. We are only interested in identifying where these two tree topologies join/merge with each other, *i.e.*, we only want to identify the joining points of S_1 and S_2 trees.

[6]Since we focus on $M = 2$, *i.e.*, only two sources S_1 and S_2, we represent the quartets (S_1, S_2, R_i, R_j) only by the receivers (R_i, R_j) for brevity.

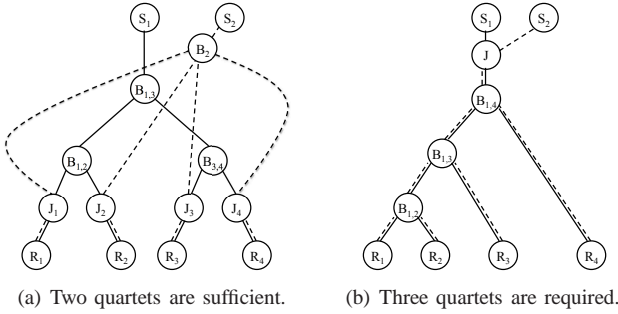(a) Two quartets are sufficient.   (b) Three quartets are required.

Fig. 3. Two example 2-by-N topologies with $N = 4$. In (a), $\frac{N}{2}$ quartets are sufficient to identify the joining points, *i.e.,* (R_1, R_2) and (R_3, R_4). In (b), more than $\frac{N}{2}$ quartets are required, *e.g.,* (R_1, R_2), (R_1, R_3), and (R_1, R_4).

by considering (R_1, R_3). It is again of type 1, which means that $J_1 \equiv J_3$ is located above $B_{1,3}$. Therefore, we go one step further and consider (R_1, R_4). Since this is also of type 1, $J_1 \equiv J_4$ lies above $B_{1,4}$. At this step, we only have a single link between S_1 and $B_{1,4}$ and thus, $J_1 \equiv J_2 \equiv J_3 \equiv J_4$ are all identified (depicted as J in Fig. 3(b)). Although there are other choices of triplets of quartets, in this topology, at least 3 quartets are required. ∎

From these examples, one can see that the lower bound of $\lceil \frac{N}{2} \rceil$ is not tight and it is not achievable in every topology. Theorem 4.1 follows from the following lemma.

Lemma 4.2: In order for an algorithm to identify all joining points for all the receivers, each receiver needs to appear in the set of quartets queried by the algorithm at least once.

Proof: Assume that there exists a receiver R_i that has not been queried in any of the quartets. We show that even with complete knowledge of all other joining points, there exist at least two possible and feasible locations for J_i, as follows.

Location 1: J_i lies on the last incoming link to R_i, *i.e.,* on the link between the parent of R_i in the S_1 tree (which from now on, we denote by $parent(R_i)$), and R_i. For example in Fig. 3(a) and Fig. 3(b), assume that $R_i = R_2$; then Location 1 would be the link $(B_{1,2}, R_2)$. This is allowed by the routing assumptions in Section III because (1) there is a unique path P_{2i}; (2) P_{2i} never merges with P_{2j}, $j \neq i$; and (3) P_{2i} merges with P_{1i} at J_i, and they continue together until they reach R_i.

Location 2: Define J_i as follows. On path P_{1i}, start at $parent(R_i)$ and move up towards S_1, until the first link that does not fully overlap with any P_{2j}, $j \neq i$. Place J_i on that link. For example in Fig. 3(a), Location 2 for J_2 would be the link $(B_{1,3}, B_{1,2})$; whereas in Fig. 3(b), it would be the link $(S_1, B_{1,4})$. This location is also allowed by the assumptions in Section III:

A1 There is a unique path P_{2i}.
A2 For every $j \neq i$, the two paths P_{2i} and P_{2j} never join after they branch. Indeed, if J_j is located above J_i on P_{1i}, then this is guaranteed by the construction of J_i. In contrast, J_j cannot be located below J_i on P_{1i} since this would imply the violation of A2 even before adding J_i.
A3 P_{2i} merges with P_{1i} at J_i and they never split.

Thus, both Location 1 and Location 2 are valid for J_i, according to the routing assumptions, and J_i cannot be uniquely identified. Therefore, R_i needs to be queried at least once. ∎

Theorem 4.1 follows from the following reasoning: each quartet involves two receivers, and thus, at least $\lceil \frac{N}{2} \rceil$ quartets are required for each receiver to appear in the set of quartets queried by the algorithm at least once.

V. A GENERALIZED BINARY SEARCH ALGORITHM

A. Background on GBS

The GBS approach has been proposed for the problem of determining a binary-valued function through a sequence of strategically selected queries, as explained in the following [35]. Consider a finite (potentially very large) collection of binary-valued functions $\mathcal{H}$, called the "hypothesis space", defined on a domain $\mathcal{X}$, called the "query space". Each $h \in \mathcal{H}$ is a mapping from $\mathcal{X}$ to $\{+1, -1\}$. Let $|\mathcal{H}|$ denote the cardinality of $\mathcal{H}$, *i.e.,* the total number of hypotheses. The functions $h \in \mathcal{H}$ are assumed to be unique, and one function, $h^* \in \mathcal{H}$, produces the correct binary labeling. h^* is assumed to be fixed but unknown. The goal is to determine h^* through as few queries from $\mathcal{X}$ as possible. Therefore, the queries need to be selected strategically in a sequential manner such that h^* is identified as quickly as possible.

It has been shown that the learning problem described above is NP-complete [41]; a practical heuristic has been proposed in the form of a greedy algorithm called Generalized Binary Search (GBS). At each step, GBS selects a query that results in the most even split of the hypotheses under consideration into two subsets, responding $+1$ and -1, respectively, to the query. The correct response to the query eliminates one of these two subsets from further consideration. The work in [35] characterizes the worst-case number of queries required by GBS in order to identify the correct hypothesis h^*. The main result of [35] indicates that under certain conditions on the query and hypothesis spaces, the query complexity of GBS (*i.e.,* the minimum number of queries required by GBS to identify h^*) is near-optimal, *i.e.,* within a constant factor of $\log_2 |\mathcal{H}|$. The constant depends on two parameters c^* and k, defined in [35], and it is desirable that they are both as small as possible.

In this section, we pose our problem in the GBS framework and use the GBS algorithm because (i) our problem is one of active learning and lends itself naturally to be posed in the GBS framework, and (ii) GBS is a principled (although not optimal) approach with provable correctness and performance guarantees [35].

B. Merging Logical Topologies in the GBS Framework

In this section, we formulate our problem within the GBS framework. Consider a set of hypotheses $\mathcal{H}$, where each hypothesis $h \in \mathcal{H}$ is a configuration that results from placing each joining point J_i on an arbitrary link in the path P_{1i} in the S_1 tree. The query space $\mathcal{X}$ is the set of all queries for all the quartets, where each query $x \in \mathcal{X}$ asks about the type of a quartet (R_i, R_j). Since in our problem, each such query x has 4 possible answers (corresponding to the 4 quartet types), we need to modify our queries to make them consistent with the binary functions in the standard GBS framework. We assume that each query x consists of 4 subqueries, each of which asks

---

Algorithm 1 GBS algorithm for identifying the joining points.

1: Let $J = [0, 0, ..., 0]$ be a vector of length N, which represents the locations of the joining points.
2: **while** $\exists 0$ in J **do**
3:      Let $wcB = [\,]$ represent the worst case benefits for all the quartets.
4:      **for** each receiver R_i **do**
5:          **for** each receiver R_j, $j > i$ **do**
6:              Let $B_{i,j}$ be the lowest common ancestor of R_i, R_j in $G_{S_1 \times \mathcal{R}}$
7:              Let $up_i \subset P_{1i}$ be the subset of P_{1i} located above $B_{i,j}$
8:              Let $up_j \subset P_{1j}$ be the subset of P_{1j} located above $B_{i,j}$
9:              Let $dn_j \subset P_{1i}$ be the subset of P_{1i} located below $B_{i,j}$
10:             Let $dn_j \subset P_{1j}$ be the subset of P_{1j} located below $B_{i,j}$
11:             $type1\_B = \frac{|up_i|}{|P_{1i}||P_{1j}|}$
12:             $type2\_B = \frac{|up_i||dn_j|}{|P_{1i}||P_{1j}|}$
13:             $type3\_B = \frac{|dn_i||up_j|}{|P_{1i}||P_{1j}|}$
14:             $type4\_B = \frac{|dn_i||dn_j|}{|P_{1i}||P_{1j}|}$
15:             $wcB.append(max([type1\_B, type2\_B, type3\_B, type4\_B]))$
16:      $selectedQuartet = wcB.index(min(wcB))$
17:      Let $selectedQuartetType$ be the type of $selectedQuartet$.
18:      **switch** $selectedQuartetType$ **do**
19:          **case** type 1 :
20:             $P_{1i} \longleftarrow up_i$
21:             $P_{1j} \longleftarrow up_j$
22:          **case** type 2 :
23:             $P_{1i} \longleftarrow up_i$
24:             $P_{1j} \longleftarrow dn_j$
25:          **case** type 3 :
26:             $P_{1i} \longleftarrow dn_i$
27:             $P_{1j} \longleftarrow up_j$
28:          **case** type 4 :
29:             $P_{1i} \longleftarrow dn_i$
30:             $P_{1j} \longleftarrow dn_j$
31:      **if** $|P_{1i}| == 1$ **then**
32:          $J_i = P_{1i}$
33:      **if** $|P_{1j}| == 1$ **then**
34:          $J_j = P_{1j}$
35: Output J.

---

whether (R_i, R_j) is of a specific type (1, 2, 3, or 4) or not; *i.e.*:

$$x = \begin{cases} \text{Is } (R_i, R_j) \text{ of type 1?} \\ \text{Is } (R_i, R_j) \text{ of type 2?} \\ \text{Is } (R_i, R_j) \text{ of type 3?} \\ \text{Is } (R_i, R_j) \text{ of type 4?} \end{cases}$$

The answer to each such subquery is binary, which is consistent with the GBS formulation. Of course, not all four subqueries are always required for a quartet; one would stop as soon as she gets the first "yes", which would reveal the type of the quartet. Note, however, that we count the number of queries (not subqueries) as the performance metric of the GBS algorithm.

Our goal is to find the target hypothesis h^*, which is the configuration that results from the correct placement of the joining points in the S_1 topology, using as few queries (*i.e.,* the knowledge of as few quartet types) as possible.[7]

Algorithm 1 describes a greedy strategy based on GBS for determining h^*. In the beginning, there are $|\mathcal{H}|$ possible hypotheses. In each step, the algorithm selects the best (*i.e.,*

---

[7]More formally, h^* answers every query, for any pair of receivers, in accordance with the true 2-by-N topology. Mathematically, h^* is a mapping from queries to $\{+1, -1\}$, not a topology itself. However, there is a bijection between all 2-by-N logical topologies and the corresponding mappings in $\mathcal{H}$, and therefore, knowing h^* is equivalent to knowing the 2-by-N topology.

maximally discriminating [35]) quartet to query as follows. By querying a quartet and learning its type, some information is obtained about the locations of two joining points. Thus, the number of feasible hypotheses, which agree with the constraints imposed by the quartets queried and learned so far, is reduced by a number, which depends on the topology in general. We call this number the *benefit* of the quartet. The best quartet to select to query is the one with maximum benefit. However, the benefit of each quartet becomes known only *after* it is queried. Thus, the algorithm considers all four possible types for every possible quartet, and focuses on the worst case benefit of that quartet, *i.e.,* the type that gives the minimum benefit. The best quartet to query is the one with maximum worst case benefit.

We denote the benefit of each type for a quartet (R_i, R_j) by $type1\_B, \cdots, type4\_B$ in Alg. 1, and define it as follows. Each quartet type limits the number of candidate edges where J_i and J_j can be located on, in the way depicted in Fig. 2. The benefit of a type for (R_i, R_j) is the ratio of the number of edges where J_i and J_j can potentially be located on after learning this type, divided by the current number of candidate edges for the locations of J_i and J_j. The worst case (minimum) benefit of (R_i, R_j) results from the type for which this ratio is maximized, and the maximum of these worst case benefits over all quartets is given by the quartet with minimum ratio.

In order to provide an analytical upper bound on the number of quartets required by Alg. 1, one can try to use the main result of [35], which indicates that Alg. 1 requires $\log_2 |\mathcal{H}|$ quartets.[8] However, we cannot compute $|\mathcal{H}|$ exactly in our problem; we can only provide a loose upper bound on that, which is $N!$.[9] Therefore, we obtain the upper bound of $\log N! \approx N \log N$ on the number of quartets required by Algorithm 1, which is loose, and much larger than the lower bound. In the next section, we evaluate the performance of Alg. 1 via simulation to obtain a better estimate of the number of quartets it requires to query in order to infer different topologies.

C. Performance Evaluation

1) Simulation Setup: We evaluate Alg. 1 in simulations over both synthetic topologies (as shown in Fig. 4) and realistic topologies (as shown in Fig. 5), and we compare it to the lower bound. The main performance metric of interest is the number of quartets queried in order to exactly infer the topology, which directly translates into measurement overhead. Additional metrics include the running time and the memory used by the algorithm, *i.e.,* the computational complexity.

For the synthetic topologies, we illustrate only the 1-by-N tree topology of S_1 in Fig. 4. We consider the star topology, "perfect" and "tall" binary trees (referring to the topologies

---

[8]This is the best case, where the constants c^* and k in [35] are both as small as possible. In practice, there is an additional constant factor for $\log_2 |\mathcal{H}|$.

[9]The bound is obtained by starting from the S_1 tree and considering all possible placements of J_i on P_{1i}, $\forall i$. Fig. 4(c) shows that there are $N \times N \times (N-1) \cdots \times 2 \cong N!$ possible such placements. In practice, the routing assumptions in Section III impose some constraints on possible J_i locations. Also, the type of each quartet may rule out some types for the other quartets. Therefore, the exact $|\mathcal{H}|$ depends on the topology and we cannot compute it.
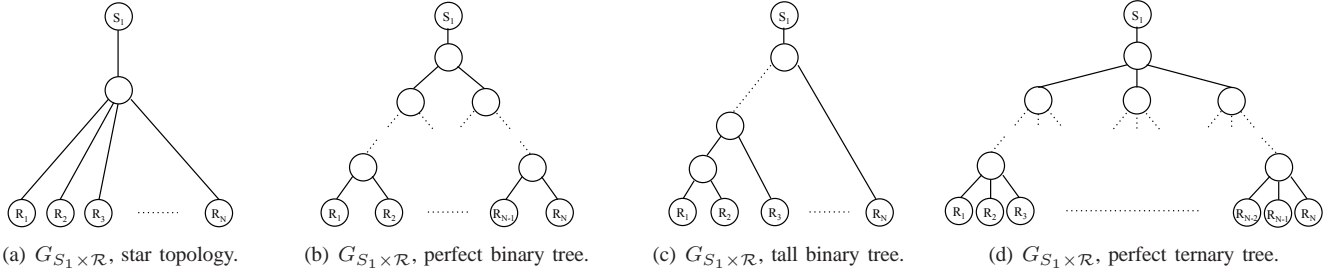
(a) $G_{S_1 \times \mathcal{R}}$, star topology.

(b) $G_{S_1 \times \mathcal{R}}$, perfect binary tree.

(c) $G_{S_1 \times \mathcal{R}}$, tall binary tree.

(d) $G_{S_1 \times \mathcal{R}}$, perfect ternary tree.

Fig. 4. Four synthetic $G_{S_1 \times \mathcal{R}}$ topologies used to evaluate the performance of Alg. 1 (the GBS approach) in simulations.

(a) A realistic 2-by-16 topology from a US University departmental LAN [3].

(b) A 2-by-16 topology generated from the Exodus topology [28].
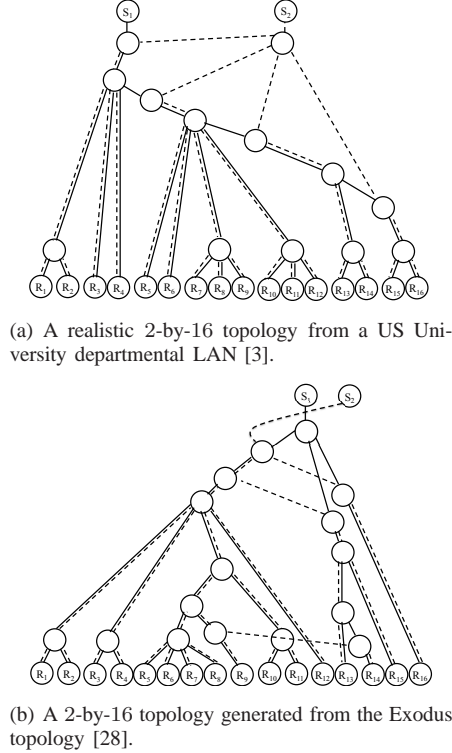
Fig. 5. Two realistic 2-by-N topologies used to evaluate the performance of Alg. 1 (GBS). The solid lines indicate the paths taken by probes from S_1 and the dashed lines indicate the paths taken by probes from S_2.

Fig. 6. Simulation results for the average number of quartets required by Alg. 1 (GBS) to infer the 2-by-N topology when $G_{S_1 \times \mathcal{R}}$ is a perfect binary tree (Fig. 4(b)) of various sizes, $N = 4, ..., 128$. The results are averaged over 100 realizations of random placements of the joining points. The standard deviation error bars (not shown) are comparable with the marker size. The figure also shows the number of quartets required by Alg. 2 (REA) and the lower bound in comparison to Alg. 1 (GBS).

depicted in Fig. 4(b) and 4(c), respectively), and perfect ternary trees, for the $G_{S_1 \times \mathcal{R}}$ tree topology. Starting from this tree, we then create a 2-by-N topology, with sources S_1 and S_2, by choosing the location of each joining point J_i (for receiver R_i) on a single logical link, selected uniformly at random, on P_{1i} in $G_{S_1 \times \mathcal{R}}$. For each $G_{S_1 \times \mathcal{R}}$ in Fig. 4, we consider 100 realizations of such random placements (resulting in different 2-by-N topologies) and report the average number of quartets required for these topologies in the next section.

For the realistic topologies, we show the complete 2-by-N topology in Fig. 5. Fig. 5(a) depicts a US University departmental LAN with 16 receivers, first used in [3]. Fig. 5(b) is a 2-by-16 directed acyclic graph (DAG), extracted from the Exodus topology, which is a large commercial ISP whose backbone map was inferred by the Rocketfuel project [28]. To generate this topology, we picked randomly two nodes of Exodus (nodes 5, 36) to be the sources, and selected all sixteen
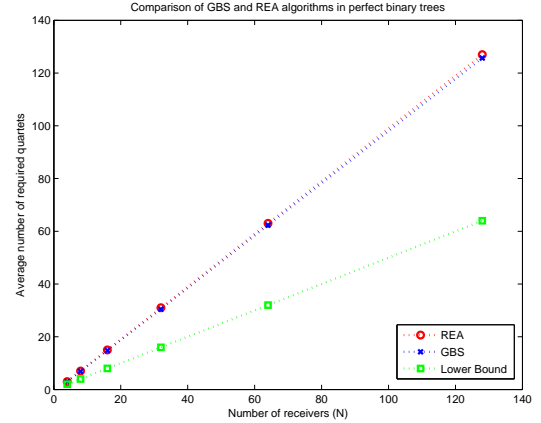
nodes to which both sources had routes to be the receivers. We then found the shortest path trees from each source to the receivers, and considered the overlap between these two trees.

Our experiments are conducted using the Python implementation of Algorithm 1, which we have made available online [42]. It takes as input any topology and returns the number of quartets required by Algorithm 1 to infer that topology. Next, we summarize the simulation results.

2) Simulation Results (for the Number of Quartets): When $G_{S_1 \times \mathcal{R}}$ is a star topology as depicted in Fig. 4(a), Alg. 1 always identifies the 2-by-N topology by querying only $\lceil \frac{N}{2} \rceil$ quartets, which is the lower bound. Therefore, it is optimal.

When $G_{S_1 \times \mathcal{R}}$ is a perfect binary tree as shown in Fig. 4(b), Alg. 1 requires different numbers of quartets, between $\frac{N}{2}$ and N, in different 2-by-N topologies. However, as shown in Fig. 6, on average, Alg. 1 requires $\sim N$ quartets.

Similar results are obtained for tall binary trees (Fig. 4(c)) and perfect ternary trees (Fig. 4(d)). Here, we omit the figures and only report the results. When $G_{S_1 \times \mathcal{R}}$ is a tall binary tree, the number of quartets required by Alg. 1 varies depending on the quartet types in different 2-by-N topologies; however, in our simulations on tall binary trees with $N > 100$ receivers, we observe that in at least 80% of the realizations, Alg. 1 requires $N - 1$ quartets. This percentage increases up to 99%
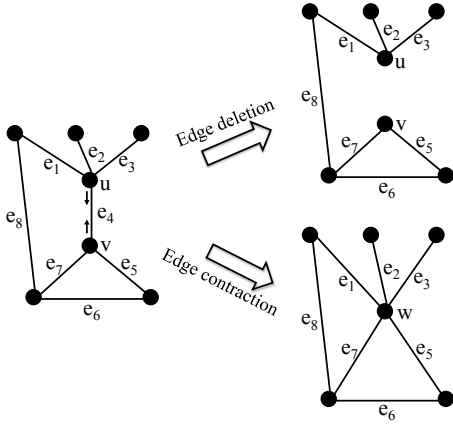
Fig. 7. Deletion and contraction of edge e_4 in a graph.

in topologies with $N < 100$ receivers. When $G_{S_1 \times \mathcal{R}}$ is a perfect ternary tree, again on average, Alg. 1 requires $N-1$ quartets, while for some topologies, it requires even more than N quartets.

For the realistic topologies in Fig. 5(a) and Fig. 5(b), Alg. 1 identifies both 2-by-16 topologies by querying 14 ($= N - 2$) quartets.

Therefore, in our simulations, we find out that Alg. 1 only performs as well as one could hope for, *i.e.,* it requires as few quartets as the lower bound, for flat $G_{S_1 \times \mathcal{R}}$ topologies, such as the star topology in Fig. 4(a). In other topologies, such as binary/ternary trees or realistic topologies, it requires many more queries, and each round of querying is extremely complex: at each step, Alg. 1 needs to calculate the worst case benefits of all the quartets, in order to pick the best one among them. In fact, the time complexity of Alg. 1 is $O(N^3)$, and its memory requirement is also high because it requires to keep track of all the benefits and the worst case benefits for all the quartets, as well as all the path updates for the location of each joining point, and so forth.

Since Alg. 1 is not very efficient in practice as described above, we propose an alternate algorithm in the next section, which is much simpler and more efficient than the GBS approach.

VI. THE RECEIVER ELIMINATION ALGORITHM

In this section, we design another greedy algorithm as an alternative to the GBS approach, called the Receiver Elimination Algorithm (REA), which requires more queries than GBS for some topologies, but each iteration is extremely simple and fast, and it scales linearly in the number of receivers. In particular, given $G_{S_1 \times \mathcal{R}}$ and the ability to query the type of any quartet, REA is able to identify all N joining points where $G_{S_2 \times \mathcal{R}}$ merges with $G_{S_1 \times \mathcal{R}}$, *i.e.,* the entire 2-by-N topology, in $N-1$ steps.

Let every edge e in $G_{S_1 \times \mathcal{R}}$ have a unique name: $label(e)$. In our algorithm, we use two operations "edge deletion" and "edge contraction", depicted in Fig. 7 and defined as follows.

Definition 1: Deleting edge (u, v), entails taking that edge out of the graph while the end-nodes u and v, and the labels

---

Algorithm 2 REA starts from $G_{S_1 \times \mathcal{R}}$, selects the quartets sequentially, queries their types, and merges them until identifying all joining points $\mathcal{J}_N$.

1: Let J be a vector of length N of edge labels, which represents the locations of the joining points.
2: **while** $|\mathcal{R}| > 1$ **do**
3:     Pick any two receivers R_i, R_j in $G_{S_1 \times \mathcal{R}}$, such that R_i and R_j are siblings; denote their parent by P.
4:     Query the type of (R_i, R_j).
5:     **switch** (R_i, R_j) **do**
6:         **case** type 1 **:**
7:             $J_i \equiv J_j$
8:             Delete R_i and edge (P, R_i).
9:             **if** outdeg(P)==1 **then**
10:                 Contract (P, R_j) into R_j.
11:         **case** type 2 **:**
12:             $J_j = label((P, R_j))$
13:             Delete R_j and edge (P, R_j).
14:             **if** outdeg(P)==1 **then**
15:                 Contract (P, R_i) into R_i.
16:         **case** type 3 **:**
17:             $J_i = label((P, R_i))$
18:             Delete R_i and edge (P, R_i).
19:             **if** outdeg(P)==1 **then**
20:                 Contract (P, R_j) into R_j.
21:          **case** type 4 **:**
22:             $J_j = label((P, R_j))$
23:             Delete R_j and edge (P, R_j).
24:             **if** outdeg(P)==1 **then**
25:                 Contract $(parent(P), P)$ into P.
26: /*There is one remaining receiver, which we call R_z.*/
27: Let $J_z = label((parent(R_z), R_z))$.
28: Output J.

---

of the remaining edges in the graph remain unchanged.

Definition 2: Contracting edge (u, v) into node w, consists of deleting that edge and merging u and v into a single node w. The labels of the remaining edges do not change (although nodes may be renamed to w).

REA is described in Alg. 2. It starts from the S_1 tree ($G_{S_1 \times \mathcal{R}}$) and proceeds by selecting one quartet to query at each step (*i.e.,* two receivers R_i, R_j to send probes to, from sources S_1, S_2). The two receivers (R_i, R_j) in the selected quartet are sibling leaves. Based on the type of the selected quartet, Alg. 2 identifies exactly one joining point in one step. It then updates $G_{S_1 \times \mathcal{R}}$ by deleting the receiver whose joining point has been identified and the last incoming edge to that receiver. That is why we call it the Receiver Elimination Algorithm. Furthermore, if a node of degree two appears in $G_{S_1 \times \mathcal{R}}$ as a result of this edge deletion, the algorithm eliminates that node by contracting the corresponding edge. The algorithm continues iteratively until there is one edge left, *i.e.,* all joining points are identified. This way, Alg. 2 identifies all joining points (where paths from S_2 to each receiver join the S_1 tree), one-by-one, proceeding from the bottom to the root of the tree. Next, we describe an illustrative example.

Example 4: Fig. 8(b)-(e) demonstrate the steps performed by REA to identify the 2-by-N topology shown in Fig. 8(a). The algorithm starts from $G_{S_1 \times \mathcal{R}}$ shown in Fig. 8(b); $e_1, ..., e_6$ are the edge labels on this tree. The algorithm first selects (R_2, R_3) and queries its type. Since the answer is type 1, the algorithm assigns $J_2 \equiv J_3$, and deletes R_2 and e_5. Since the degree of $B_{2,3}$ becomes 2, the algorithm contracts e_6 into R_3.

(a) The $G_{\mathcal{S} \times \mathcal{R}}$ topology, which we want to identify.

(b) $G_{S_1 \times \mathcal{R}}$ (T_4). (R_2, R_3) is of type 1; thus $J_2 \equiv J_3$.

(c) T_3. (R_1, R_3) is of type 4; thus J_3 is identified on e_3.

(d) T_2. (R_1, R_4) is of type 3; thus J_1 is identified on e_2.

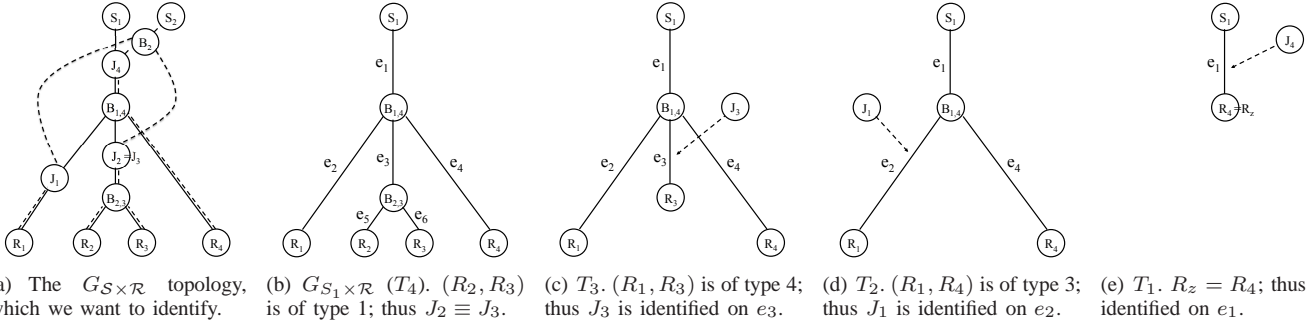(e) T_1. $R_z = R_4$; thus J_4 is identified on e_1.

Fig. 8. The steps (b), (c), (d), and (e), performed by Alg. 2 (REA) to identify the 2-by-N topology in (a). The output of the algorithm is $J = [e_2, e_3, e_3, e_1]$.

In the second step shown in Fig. 8(c), REA selects two sibling leaves (R_1, R_3), randomly out of three possible pairs of siblings, and queries its type. Since it is type 4, the algorithm identifies J_3 on e_3 (which, together with the previous step, means that J_2 is also identified). It also deletes R_3 and e_3. There is no contraction in this step as $B_{1,4}$'s degree is > 2.

In the third step shown in Fig. 8(d), (R_1, R_4) is selected and queried; it is of type 3. Therefore, the algorithm identifies J_1 on e_2, deletes R_1 and e_2, and contracts e_4 into R_4. Since there is only one receiver left, there are no more quartets to query; thus the algorithm exits the while loop and proceeds to the last step (line 26). For $R_z = R_4$, the algorithm identifies J_4 on e_1, as shown in Fig. 8(e). The identified joining points agree with the real locations in $G_{\mathcal{S} \times \mathcal{R}}$ topology in Fig. 8(a), which demonstrates the correctness of the algorithm. ∎

A. Properties of REA

Let $T_N = G_{S_1 \times \mathcal{R}}$ denote the logical tree from S_1 to all N receivers, which we assume to be known. In this section, we use the notation T_N to emphasize that this initial tree $G_{S_1 \times \mathcal{R}}$ contains N receivers. After each iteration through the while loop in Alg. 2, one receiver is deleted. We write T_k to denote the tree (rooted at S_1) obtained at the end of iteration $(N-k)$, at which point there are k receivers remaining. Let $\mathcal{J}_k$ denote the set of joining points, which still remain to be identified after iteration $(N-k)$, i.e., one for each remaining receiver.

Proposition 6.1: Let T_k and $\mathcal{J}_k$ be given. The next iteration of Alg. 2 (lines $3-25$) produces T_{k-1} and $\mathcal{J}_{k-1}$, which satisfy the following properties:

1) The S_1 topology is still a logical tree, and it has $k-1$ receivers (i.e., one receiver and its corresponding edge are deleted from T_k). Therefore, we denote it by T_{k-1}.

2) One joining point has been identified; therefore, the algorithm has $k-1$ more joining points in $\mathcal{J}_{k-1}$ to identify.

3) All joining points in $\mathcal{J}_{k-1}$ are located on edges in T_{k-1}.

Proof: These properties follow directly from the operations performed by one step of Alg. 2:

1) In each iteration, a single receiver is eliminated from the tree. Consequently, the only node that can possibly have degree two (or out-degree one) after deleting the receiver is its parent, P. However, after each deletion, Alg. 2 tests to see if P has out-degree 1, and if it does, then an additional contraction is performed so that the resulting tree, T_{k-1}, is still logical.

2) When (R_i, R_j) is of type 2, 3, or 4, we can see in lines 12, 17, and 22 of the algorithm, respectively, that one joining point is identified. When (R_i, R_j) is of type 1, line 7 assigns to R_i, the same joining point as R_j's. Then, in line 8, R_i is deleted so that we do not create a loop by assigning J_i again to J_j later. Also, J_j eventually becomes identified, either in one of the other types (2, 3, or 4) in the while loop, or in the last line of the algorithm. Thus, we have $\mathcal{J}_{k-1}$ after one step.

3) Alg. 2 changes T_k by 2 processes: edge deletion and edge contraction. We show that neither deletion nor contraction can eliminate an edge in T_k that contains a joining point in $\mathcal{J}_{k-1}$.

Deletion: Alg. 2 is constructed such that any edge deleted from the S_1 tree contains either no joining point (if (R_i, R_j) is of type 1) or exactly one joining point, corresponding to the receiver being removed along with that edge (if (R_i, R_j) is of type 2, 3, or 4).

Contraction: An edge is contracted only when it does not contain any joining point, neither for R_i and R_j (see lines $9-10$ for type 1, lines $14-15$ for type 2, lines $19-20$ for type 3, and lines $24-25$ for type 4), nor for any other receivers (since (R_i, R_j) are sibling leaves, the contracted edge cannot contain any joining point for any other receiver.[10]). ∎

The following theorem establishes the correctness and complexity of Algorithm 2 (REA).

Theorem 6.2: REA terminates in N steps and correctly identifies all N joining points after querying $N-1$ quartets.

Proof: The proof is via induction. In the beginning, $T_N = G_{S_1 \times \mathcal{R}}$ is a logical tree and according to Corollary 1 in [1], the joining points are identifiable using sufficient quartets. Our inductive step is one iteration of the while loop. First, note that there exist two sibling receivers at every step: it is enough to pick one of the lowest receivers (i.e., a receiver with the largest distance from the source); it will always have a sibling because of the logical tree topology. The algorithm queries one quartet per step, identifies one joining point per step, and at the end of the step, it preserves properties 1, 2, and 3. The while loop terminates in $N-1$ iterations and there is one additional step for R_z after the loop (which does not use any quartet). Therefore, the algorithm terminates in N steps, and correctly identifies all N joining points by querying exactly $N-1$ quartets. ∎

[10]Algorithm 2 selects *sibling* receivers R_i and R_j at each step. Therefore, there are only two potential candidates for the joining points that can be identified at this step: J_i and J_j.

Discussion. An important observation is that the $N-1$ quartets are not known a priori, but are easily selected in a sequential way, as needed; this makes REA easy to implement in practice using active probing. Another observation is about the running time: exactly $N-1$ quartets need to be queried (by sending sets of probes). This is much less than the $\binom{N}{2}$ possible quartets queried by a brute-force approach [1, 3], but higher than the lower bound on the number of required quartets by any algorithm ($\lceil \frac{N}{2} \rceil$, Theorem 4.1). Therefore, REA is not optimal, but it is simple, efficient, and provably correct. The next section compares the performance of REA to GBS in different topologies.

B. Comparison to GBS

In Section V-C, we evaluated the performance of Alg. 1 (GBS) in simulations over both synthetic topologies of Fig. 4 and realistic topologies of Fig. 5. In this section, we compare the performance of Alg. 2 (REA) against Alg. 1 (GBS) and the lower bound, over the same topologies. The performance metrics of interest include the number of quartets queried in order to exactly infer the topology, *i.e.,* the measurement overhead, as well as the running time and the memory used by each algorithm.

1) The Number of Quartets: When $G_{S_1 \times \mathcal{R}}$ is a star topology as in Fig. 4(a), we saw in Section V-C that GBS is optimal and requires only $\lceil \frac{N}{2} \rceil$ quartets. Therefore, it performs better than REA, which requires $N-1$ quartets.

On the other hand, when $G_{S_1 \times \mathcal{R}}$ is a perfect binary tree as in Fig. 4(b), we can see in Fig. 6 that on average, REA performs very close to GBS, while GBS is much more complex than REA. Similar results are obtained for tall binary trees (Fig. 4(c)) and perfect ternary trees (Fig. 4(d)). As described in Section V-C, for both $G_{S_1 \times \mathcal{R}}$ topologies, on average, REA performs close to GBS, and for some topologies, GBS requires even more than N quartets.

For the realistic 2-by-16 topologies in Fig. 5(a) and Fig. 5(b), we saw in Section V-C that GBS requires $N-2 = 14$ quartets, while REA requires $N-1 = 15$ quartets.

Therefore, one can see that GBS only requires significantly fewer quartets than REA for flat $G_{S_1 \times \mathcal{R}}$ topologies, such as the star topology in Fig. 4(a). In other topologies, such as binary/ternary trees or realistic topologies, REA is preferred over GBS, because it is much simpler and it identifies the joining points using the same number of quartets (or even fewer quartets in large topologies) as GBS (*i.e.,* $N-1$).

2) Time and Space Complexity: The time complexity of REA ($O(N)$) is significantly lower than that of GBS ($O(N^3)$). The reason is that at each step, REA only needs to select a pair of sibling receivers (any of them will do); while GBS calculates the worst case benefits of all the quartets, in order to pick the best one among them, which takes much longer. As an example, for a single realization of our simulations when $G_{S_1 \times \mathcal{R}}$ is a perfect binary tree with 128 receivers, the running time of REA is only < 1 second, while that of GBS is 19 seconds. This is a big difference when we consider a large number of realizations as described in Section V-C.

The memory requirement of REA is also much lower than that of GBS. The reason is that REA only requires to store the (modified version of the) graph at each step; while GBS requires to keep track of all the benefits and the worst case benefits for all the quartets, all the path updates for the location of each joining point, and so forth.

VII. EXTENSIONS

In this section, we briefly outline the possible extensions to the active learning algorithms we have discussed so far.

A. Extension to M-by-N Topologies

So far, we have focused on inferring a 2-by-N topology, which is a special but important case. M-by-N topologies can be inferred by merging the tree topologies of the remaining $M-2$ sources to this 2-by-N topology, one source at a time. Assume that we have inferred a k-by-N topology, $2 \le k < M$. To add the $(k+1)^{th}$ source, we need to identify each joining point of the new source, S_{k+1}, and any one of the k sources in the current topology, S_i, $1 \le i \le k$, for each receiver, on a single logical link in the k-by-N topology (defined by all the branching points). Therefore, we need to apply REA (or GBS) to S_{k+1} and any one (in the best case) or all (in the worst case) of the current k sources. Therefore, for example using REA, the number of quartets required to identify the M-by-N topology is between $(M-1)(N-1)$ and $\binom{M}{2}(N-1)$.

B. Extension to Noisy Case

So far, we have considered the noiseless scenario, where the answer to each query is the correct quartet type. One can extend the algorithms to deal with noisy queries, using the two approaches proposed in [35]. The first one is a simple solution that applies to both GBS and REA; it repeats the query multiple times and considers the majority vote as the answer to that query. The second approach is more sophisticated and fits naturally in the GBS framework.[11] It assigns weights to each hypothesis using a probability distribution. The initial weighting is uniform, but it gets updated after each query. The update naturally boosts the probability measure of the hypotheses that agree with the answer to the query. At the end, the hypothesis with the largest weight is selected. We can adopt this approach for the GBS algorithm by incorporating the probability measures in the path updates and in computing the benefits. Using this approach, GBS can handle the noisy queries more naturally than REA. The query complexity and the probability of error for both approaches have been analyzed in [35].

VIII. CONCLUSION

Although active topology inference is a well-studied problem, to the best of our knowledge, this paper is the first to focus on efficient merging algorithms. We formulate the problem as multiple hypothesis testing and develop an active learning algorithm based on GBS. We also propose an efficient Receiver Elimination Algorithm that queries only $N-1$ quartets, which is much less than the $\binom{N}{2}$ possible quartets. Furthermore,

---

[11]A similar solution for REA would be to perform the deletions and contractions probabilistically.

comparing it to the GBS algorithm in simulations, we find out that the simple REA is near-optimal, and comparable to the GBS approach in terms of the number of queries (thus measurement bandwidth), while having much lower time and space complexity. Therefore, it is preferable for all practical purposes.

REFERENCES

[1] M. Rabbat, M. Coates, and R. Nowak, "Multiple source internet tomography," *IEEE JSAC*, vol. 24, no. 12, pp. 2221–2234, Dec. 2006.

[2] M. Rabbat, R. Nowak, and M. Coates, "Multiple source, multiple destination network tomography," in *Proc. of IEEE Infocom*, Hong Kong, March 2004.

[3] M. Coates, M. Rabbat, and R. Nowak, "Merging logical topologies using end-to-end measurements," in *Proc. of ACM IMC*, Miami, FL, Oct. 2003.

[4] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: Recent developments," *Statistical Science*, vol. 19, no. 3, pp. 499–517, 2004.

[5] A. Anandkumar, A. Hassidim, and J. Kelner, "Topology discovery of sparse random graphs with few participants," in *Proc. of ACM Sigmetrics*, San Jose, CA, June 2011.

[6] A. Krishnamurthy and A. Singh, "Robust multi-source network tomography using selective probes," in *Proc. of IEEE Infocom*, Orlando, FL, March 2012.

[7] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *Proc. of IEEE Infocom*, New York, NY, March 1999, pp. 353–360.

[8] N. Duffield, J. Horowitz, and F. L. Presti, "Adaptive multicast topology inference," in *Proc. of IEEE Infocom*, Anchorage, AK, April 2001.

[9] N. Duffield, J. Horowitz, F. Presti, and D. Towsley, "Multicast topology inference from measured end-to-end loss," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 26–45, Jan. 2002.

[10] M. Coates, R. Castro, M. Gadhiok, R. King, Y. Tsang, and R. Nowak, "Maximum likelihood network topology identification from edge-based unicast measurements," in *Proc. of ACM Sigmetrics*, Marina Del Rey, CA, June 2002.

[11] N. G. Duffield and F. L. Presti, "Network tomography from measured end-to-end delay covariance," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 978–992, Dec. 2004.

[12] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak, "Toward the practical use of network tomography for internet topology discovery," in *Proc. of IEEE Infocom*, San Diego, CA, March 2010.

[13] J. Ni, H. Xie, S. Tatikonda, and Y. R. Yang, "Efficient and dynamic routing topology inference from end-to-end measurements," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, Feb. 2010.

[14] J. Ni and S. Tatikonda, "Network tomography based on additive metrics," *IEEE Transactions on Information Theory*, vol. 57, no. 12, pp. 7798–7809, Dec. 2011.

[15] M. Choi, V. Tan, A. Anandkumar, and A. Willsky, "Learning latent tree graphical models," *Journal of Machine Learning Research*, vol. 12, pp. 1771–1812, May 2011.

[16] P. Sattari, A. Markopoulou, and C. Fragouli, "Multiple source multiple destination topology inference using network coding," in *Proc. of NetCod*, EPFL, Lausanne, Switzerland, June 2009.

[17] P. Sattari, C. Fragouli, and A. Markopoulou, "Active topology inference using network coding," *Elsevier PHYCOM, Special Issue on Network Coding and its Applications to Wireless Communications*, vol. 6, pp. 142–163, March 2013.

[18] I. Cunha, R. Teixeira, N. Feamster, and C. Diot, "Measurement methods for fast and accurate blackhole identification with binary tomography," in *Proc. of ACM IMC*, Chicago, Illinois, Nov. 2009.

[19] E. Katz-Bassett, H. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson, "Studying black holes in the internet with hubble," in *Proc. of USENIX NSDI*, San Francisco, CA, April 2008.

[20] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "ispy: Detecting ip prefix hijacking on my own," in *Proc. of ACM SIGCOMM*, Seattle, WA, August 2008.

[21] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[22] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 131–145, Dec. 2001.

[23] C. Fragouli, A. Markopoulou, and S. Diggavi, "Topology inference using network coding," in *Proc. of Allerton Conference*, Monticello, IL, Sept. 2006.

[24] G. Sharma, S. Jaggi, and B. Dey, "Network tomography via network coding," in *Proc. of Information Theory and Applications (ITA) Workshop*, San Diego, CA, Feb. 2008.

[25] B. Huffaker, D. Plummer, D. Moore, and K. Claffy, "Topology discovery by active probing," in *Proc. of SAINT*, Nara, Japan, Jan. 2002.

[26] B. Cheswick, H. Burch, and S. Branigan, "Mapping and visualizing the internet," in *Proc. of USENIX Annual Technical Conference (ATC)*, San Diego, CA, June 2000.

[27] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery," in *Proc. of IEEE Infocom*, Tel-Aviv, Israel, March 2000.

[28] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *Proc. of ACM Sigcomm*, Pittsburgh, PA, Aug. 2002.

[29] M.-F. Shih and A. O. Hero, "Hierarchical inference of unicast network topologies based on end-to-end measurements," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 1708–1718, 2007.

[30] Y. Tsang, M. Coates, and R. D. Nowak, "Network delay tomography," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2125–2136, 2003.

[31] M.-F. Shih and A. O. Hero, "Unicast-based inference of network link delay distributions with finite mixture models," *IEEE Transactions on Signal Processing*, vol. 51,

no. 8, pp. 2219–2228, 2003.

[32] K. Deng, Y. Li, W. Zhu, Z. Geng, and J. S. Liu, "On delay tomography: Fast algorithms and spatially dependent models," *IEEE Transactions on Signal Processing*, vol. 60, no. 11, pp. 5685–5697, 2012.

[33] G. Liang and B. Yu, "Maximum pseudo likelihood estimation in network tomography," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2043–2053, 2003.

[34] A. Chen, J. Cao, and T. Bu, "Network tomography: Identifiability and fourier domain estimation," *IEEE Transactions on Signal Processing*, vol. 58, no. 12, pp. 6029–6039, 2010.

[35] R. D. Nowak, "The geometry of generalized binary search," *IEEE Transactions on Information Theory*, vol. 57, no. 12, pp. 7893–7906, Dec. 2011.

[36] S. Dasgupta, "Coarse sample complexity bounds for active learning," in *Proc. of Neural Information Processing Systems (NIPS)*, Vancouver, B.C., Canada, Dec. 2005.

[37] R. Castro and R. Nowak, "Minimax bounds for active learning," *IEEE Transactions on Information Theory*, vol. 54, no. 5, pp. 2339–2353, May 2008.

[38] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[39] J. Pearl and M. Tarsi, "Structuring causal trees," *Journal of Complexity*, 1986.

[40] P. Sattari, M. Kurant, A. Anandkumar, A. Markopoulou, and M. Rabbat, "Active learning of multiple source multiple destination topologies," in *Proc. of IEEE CISS*, Johns Hopkins University, Baltimore, MD, March 2013.

[41] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, May 1976.

[42] P. Sattari, "Python implementations of the gbs and rea algorithms," http://odysseas.calit2.uci.edu/doku.php/public:pegah-software, July 2012.