Active Learning of Multiple Source Multiple Destination Topologies

Pegah Sattari, Maciej Kurant, Animashree Anandkumar, Athina Markopoulou
Department of EECS, University of California, Irvine
{psattari, mkurant, a.anandkumar, athina}@uci.edu
Michael Rabbat
Dept. of ECE, McGill University
michael.rabbat@mcgill.ca

Abstract—We consider the problem of inferring the topology of an $M$-by-$N$ network by sending probes between $M$ sources and $N$ 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. 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 1-by-$N$, so as to exactly identify the 2-by-$N$ topology, and optimize a number of performance metrics including measurement traffic, time complexity, and memory usage. We provide a lower bound, $\left(\frac{N^2}{2}\right)$, on the number of 2-by-2’s required by any active learning algorithm and we also propose a greedy algorithm that is near-optimal and efficient in practice. It 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-$N$. The algorithm requires exactly $N - 1$ steps, which is much less than all $\left(\binom{N}{2}\right)$ possible 2-by-2’s, and it correctly identifies the 2-by-$N$ topology.

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/received between sets of sources/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 and 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/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 a principled approach to design an efficient algorithm 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.

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 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, 18]) and that the topology of a quartet component can be queried and learned, if so desired (e.g., using end-to-end probes and some of the methods in [1, 2, 16, 17, 19–23]1). 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-$N$ and optimize several performance metrics including measurement bandwidth, merging complexity and memory usage. Our contributions are as follows:

1) We provide a lower bound of $\left(\frac{N^2}{2}\right)$ 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 design an efficient merging algorithm that follows a greedy bottom-up approach and provably identifies the 2-by-$N$ by querying exactly $N - 1$ quartets. From the active probing

This work was supported by AFOSR grant FA9550-10-1-0310.
Pegah Sattari and Maciej Kurant were with the EECS Department and Calit2 at UC Irvine when this research was conducted.

1 Other techniques may also be developed in the future as this is an active research area. However, this is out of the scope of this paper (see Section III).
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. We show that the algorithm exactly identifies the topology and is very efficient in terms of running time and memory usage. We also show that the algorithm is near-optimal in terms of active measurement bandwidth. Therefore, it is 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 an efficient bottom-up algorithm and analyzes its correctness and performance. Section VI discusses possible extensions. Section VII 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.

One family of techniques relies on cooperation of nodes in the middle of the network, and uses traceroute [20–23] measurements to collect the ids of nodes along paths. However, some nodes may not respond and nodes often have multiple network interfaces (ids). Thus, traceroute-based methods must deal with missing/incomplete data and alias problems.

Unlike traceroute, tomographic approaches do not rely on responses from intermediate nodes, but only on end-to-end measurements. A survey of network tomography can be found in [4]. 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 2 sources and packet arrival order measurements at the 2 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 with the first one. The resulting M-by-N 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 internal nodes are equipped with multicast and network coding capabilities. We built on [1] and extended it, using network coding at internal nodes to deterministically distinguish among all possible quartet topologies, which was not possible before.

While in [16, 17], we focused on inferring quartets fast and accurately, here we assume that any quartet can be queried and learned, and focus on efficiently selecting and merging 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.

Topology inference problems have also been studied in the context of phylogenetic trees [24, 25]. [6] built on [25] and proposed robust algorithms for multiple source tree topology inference. [5] inferred the topology of sparse random graphs using end-to-end measurements between a few 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).

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 \( S = \{S_1, \ldots, S_M\} \) and \( N \) receivers \( R = \{R_1, \ldots, R_N\} \). We denote this \( M \)-by-\( N \) topology by \( G_{S \times R} \). Note that \( G_{S \times R} \) 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 \([1]\):^2

A1 For every source \( S_i \) and every receiver \( R_j \), there is a unique path \( P_{ij} \).
A2 Two paths \( P_{ij} \) and \( P_{ik}, \ j \neq k \), branch at a joining 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\(^3\), 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 \( G_{S \times R} \) \( S = \{S_1, S_2\} \); an \( M \)-by-\( N \) topology, \( S = \{S_1, \ldots, S_M\} \), can then be constructed by merging smaller structures, as we describe in Section VI.

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

---

^2These 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.

^3A 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.
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 $R_2$ 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 ($B_{1,2} \equiv B_{2,2}$). However, the other three types (2, 3, 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 $R_1, R_2$, a set of probes must be sent from $S_1, S_2$ to $R_1, R_2$. 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 all four types exploiting multicast and network coding); [19] (which can exactly infer the topology of a super-source to two receivers using network coding); traceroute [20–23] from the two sources to the two receivers; or other techniques that may be developed in the future, since this 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., 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 $B_{1,2} \equiv B_{2,2}$, 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 $G_{S_1 \times R}$ (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, 18]). 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) $G_{S_1 \times R}$ and (ii) the ability to query the quartet type between $S_1, S_2$, and any two receivers $R_1, R_2$, our goal is to identify all joining points, $J_N = \{J_1, J_2, ..., J_N\}$, where the paths from $S_2$ to each receiver join the tree describing paths from $S_1$ to the same set of receivers. Identifying a joining point $J_i$ (for receiver $R_i$) means locating $J_i$ on a single logical link, between two branching points on $G_{S_1 \times R}$. 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., $J_3$ 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 $G_{S_1 \times R}$ 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 $S_1, S_2$), 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 infer the 2-by-$N$. It clearly depends on the topology we want to infer and serves as a baseline for the performance of the proposed algorithm.

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

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 needed to identify $S_1$ and $S_2$, we represent the quartets $(S_1, S_2, R_1, R_2)$ only by the receivers $(R_1, R_2)$ for brevity.
all 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 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 \(B_{1,2}\). But there is more than a single link above \(B_{1,2}\) thus we continue by considering \((R_1, R_3)\). It is again of type 1, which means that \(J_1 \equiv J_3\) is located above \(B_{1,3}\). Thus, we go one step further and consider \((R_1, R_4)\). Since this is also 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. In 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 \(S_i\) 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 \(R_i\) is located above \(B_{1,2}\); 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 \((S_1, B_{1,1})\). 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 \(J_i\).

![Fig. 3](image-url) Two example 2-by-\(N\) topologies with \(N = 4\). In (a), \(\binom{4}{2}\) quartets are sufficient to identify the joining points, i.e., \((R_1, R_2)\) and \((R_3, R_4)\). In (b), more than \(\binom{6}{2}\) quartets are required, e.g., \((R_1, R_2), (R_1, R_3)\), and \((R_1, R_4)\).

![Fig. 4](image-url) Deletion and contraction of edge \(e_4\) in a graph.

**Algorithm 1** Bottom-up merging algorithm: it starts from \(G_{S_1 \times \mathbb{R}}\), selects the quartets sequentially, queries their types, and merges them until identifying all joining points \(J_N\).

1: Let \(J\) be a vector of length \(N\) of edge labels, which represents the locations of the joining points.
2: while \(|R| > 1\) do
3: Pick any two receivers \(R_i, R_j\) in \(G_{S_1 \times \mathbb{R}}\) 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 \((P, R_i)\) into \(R_i\).
11: case type 2:
12: \(J_i = label((P, R_i))\)
13: delete \(R_i\) and edge \((P, R_i)\).
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 \((parent(P), P)\) into \(P\).
21: case type 4:
22: \(J_i = label((P, R_i))\)
23: delete \(R_i\) and edge \((P, R_i)\).
24: if outdeg\((P)\)=\(1\) then
25: contract \((parent(P), P)\) into \(P\).
26: /*There is one remaining receiver, which we call \(R_{z,t}\)*/
27: let \(J_z = label((parent(R_{z,t}), R_{z,t}))\).
28: output \(J\).

**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 BOTTOM-UP GREEDY ALGORITHM**

In this section, we design a greedy algorithm that given \(G_{S_1 \times \mathbb{R}}\), and the ability to query the type of any quartet, it is able to identify all \(N\) joining points where \(G_{S_2 \times \mathbb{R}}\) merges with \(G_{S_1 \times \mathbb{R}}\), i.e., the entire 2-by-\(N\) topology, in \(N-1\) steps.

Let every edge \(e\) in \(G_{S_1 \times \mathbb{R}}\) have a unique name: \(label(e)\). In our algorithm, we use two operations “edge deletion” and...
“edge contraction”, depicted in Fig. 4 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 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 \(w\)).

The algorithm is described in Alg. 1. It starts from the \(S_1\) tree \(G_{S_1 \times R}\) and proceeds by selecting one quartet to query at each step (i.e., 2 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. 1 identifies exactly one joining point in one step. It then updates \(G_{S_1 \times R}\) by deleting the receiver whose joining point has been identified and the last incoming edge to that receiver. Furthermore, if a node of degree two appears in \(G_{S_1 \times R}\) 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. 1 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. 5(b)-(e) demonstrate the steps performed by Alg. 1 to identify the 2-by-4 topology shown in Fig. 5(a). The algorithm starts from \(G_{S_1 \times R}\) shown in Fig. 5(b); \(e_1, \ldots, e_6\) 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 \(R_3\).

In the second step shown in Fig. 5(c), Alg. 1 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. 5(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 \(R_2 = R_4\), the algorithm identifies \(J_4\) on \(e_1\), as shown in Fig. 5(e). The identified joining points agree with the real locations in \(G_{S_1 \times R}\) topology in Fig. 5(a), which demonstrates the correctness of the algorithm.

### A. Properties of Algorithm 1

Let \(T_N = G_{S_1 \times R}\) 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 \(G_{S_1 \times R}\) contains \(N\) receivers. After each iteration through the while loop in Alg. 1, 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 \(J_k\) denote the set of joining points, which still remain to be identified after iteration \((N-k)\), i.e., one for each remaining receiver.

**Proposition 5.1:** Let \(T_k\) and \(J_k\) be given. The next iteration of Alg. 1 (lines 3–25) produces \(T_{k-1}\) and \(J_{k-1}\), 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 \(J_{k-1}\) to identify.

3. All joining points in \(J_{k-1}\) are located on edges in \(T_{k-1}\).

**Proof:** These properties follow directly from the operations performed by one step of Alg. 1:

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. 1 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 \(J_{k-1}\) after one step.

3. Alg. 1 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 \(J_{k-1}\).

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

Fig. 5. The steps (b), (c), (d), and (e), performed by Alg. 1 to identify the 2-by-4 topology in (a). The output of the algorithm is \(J = [e_2, e_3, e_4, e_1]\).
(since \((R_i, R_j)\) are sibling leaves, the contracted edge cannot contain any joining point for any other receiver.\(^5\))

The following theorem establishes the correctness and complexity of Algorithm 1.

**Theorem 5.2:** Alg. 1 terminates in \(N\) steps and correctly identifies all \(N\) joining points after querying \(N - 1\) quartets.

**Proof:** The proof is by induction. In the beginning, \(T_N = G_{S_1 \times R}\) 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 \(R_x\) after the loop (which does not use any quartet). Thus, the algorithm terminates in \(N\) steps, and correctly identifies all \(N\) joining points by querying exactly \(N - 1\) quartets.

**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 Alg. 1 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 \(\left(\frac{N}{2}\right)\). Theorem 4.1. Thus, Alg. 1 is not optimal, but it is simple, efficient, and provably correct. Alg. 1 is also efficient in terms of memory requirements, since it only stores the (modified version of the) graph at each step. We have made the Python implementation of Alg. 1 available online [26].

In our technical report [27], we have also formulated the problem as multiple hypothesis testing and developed an active learning algorithm based on Generalized Binary Search (GBS) [28]. We then compared Alg. 1 against the GBS baseline. The results show that Alg. 1 is comparable to GBS in terms of the number of queries, while having much lower time and space complexity; therefore, it is preferable for all practical purposes.

VI. 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 one source-rooted tree topology at a time. Assume that we have inferred a \(k\)-by-N topology, \(2 \leq k < M\). To add the \((k + 1)\)th source, we need to identify each joining point of \(S_{k+1}\) and \(S_i, 1 \leq i \leq k\), 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 Alg. 1 to \(S_{k+1}\) and any one (in the best case) or all (in the worst case) of the current \(k\) sources. Therefore, using Alg. 1, the number of quartets required to identify the \(M\)-by-N topology is between \((M - 1)(N - 1)\) and \(\binom{M}{2}(N - 1)\).

VII. 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 propose a greedy bottom-up approach that queries only \(N - 1\) quartets, which is much less than \(\binom{N}{2}\) possible quartets. This simple algorithm is near-optimal in terms of the number of queries (thus measurement bandwidth), and it has very low time and space complexity; therefore, it is recommended for practical implementation.

In the future, it would be interesting to extend the algorithm to deal with noisy queries (e.g., by repeating the query multiple times), and also to compare the algorithm against the optimal, computed using dynamic programming, which is both challenging to formulate and would have exponential complexity.

**References**


