Robust and Efficient Monitor Placement for Network Tomography in Dynamic Networks

Ting He, *Senior Member, IEEE*, Athanasios Gkelias, *Senior Member, IEEE*, Liang Ma, Kin K. Leung, *Fellow, IEEE*, Ananthram Swami, *Fellow, IEEE*, and Don Towsley, *Life Fellow, IEEE, Fellow, ACM*

***Abstract*— We consider the problem of placing the minimum number of monitors in a dynamic network to identify additive link metrics from path metrics measured along cycle-free paths between monitors. Our goal is robust monitor placement, i.e., the same set of monitors can maintain network identifiability under topology changes. Our main contribution is a set of monitor placement algorithms with different performance-complexity tradeoffs that can simultaneously identify multiple topologies occurring during the network lifetime. In particular, we show that the optimal monitor placement is the solution to a generalized hitting set problem, for which we provide a polynomial-time algorithm to construct the input and a greedy algorithm to select the monitors with logarithmic approximation. Although the optimal placement is NP-hard in general, we identify non-trivial special cases that can be solved efficiently. Our secondary contribution is a dynamic triconnected decomposition algorithm to compute the input needed by the monitor placement algorithms, which is the first such algorithm that can handle edge deletions. Our evaluations on mobility-induced dynamic topologies verify the efficiency and the robustness of the proposed algorithms.**

***Index Terms*— Network tomography, monitor placement, dynamic graph decomposition.**

I. Introduction

NETWORK tomography refers to the methodology of inferring internal performance metrics (e.g., link delays/losses) of a communication network from external measurements taken between nodes with monitoring capabilities, referred to as *monitors*. Since its introduction [2], network tomography has attracted significant interest in the research community as a promising alternative to the approach of direct measurement. Traditionally, network monitoring systems rely on diagnostic tools such as *traceroute*, *pathchar* [3], and *Network Characterization Service (NCS)* [4] to directly measure the performance of individual links via active probes, but these techniques suffer from high measurement overhead and lack of cooperation from internal nodes. In contrast, tomography-based monitoring only requires cooperation of monitors and can utilize passive measurements from data packets to reduce the need of active probes [5].

A major challenge in applying network tomography is the lack of uniqueness in the inferred link metrics. For example, consider the inference of link metrics that are *additive* (e.g., delays, jitters, log of packet delivery ratio). Network tomography infers such link metrics by solving a system of linear equations, where the unknown variables are the link metrics, and each measurement path provides an equation that relates the metrics of traversed links to the end-to-end measurement on this path. From linear algebra, we know that the system has a unique solution if and only if the number of *linearly independent* measurement paths equals the number of links. However, past experience shows that without careful design, it is frequently impossible to uniquely determine all the link metrics from path measurements [6]–[8].

This problem, known as the *lack of identifiability*, has been recognized in the literature, where several solutions have been proposed to ensure identifiability in a network with a fixed topology by carefully placing the monitors [9], [10]. While the fixed-topology assumption is valid in wired networks, applying network tomography to wireless networks faces the additional challenge that the network topology may vary at runtime due to factors such as node mobility, node activation/deactivation, and channel variation. While a straightforward solution is to handle the changes *reactively* by computing a new monitor placement after each topology change, such a solution can lead to frequent reconfigurations and instability in the monitoring system. To maintain seamless monitoring in such dynamic networks, it is desirable to have a monitor placement strategy that can handle topology changes *proactively*.

In this paper, we aim to develop monitor placement algorithms that are *robust* to topology changes. To this end, we leverage existing works on predicting topology changes. For example, we can predict topology changes based on models of node mobility [11], [12], patterns of link failures [13],

Manuscript received March 22, 2016; revised September 28, 2016 and December 5, 2016; accepted December 16, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Y. Eun. Date of publication January 31, 2017; date of current version June 14, 2017. This work was supported in part by the U.S. Army Research Laboratory, and in part by the U.K. Ministry of Defence under Grant W911NF-06-3-0001. This paper was presented at the IEEE INFOCOM 2016.

T. He is with The Pennsylvania State University, University Park, PA 16802 USA (e-mail: tzh58@psu.edu).

A. Gkelias and K. K. Leung are with the Imperial College, London, U.K. (e-mail: a.gkelias@imperial.ac.uk; kin.leung@imperial.ac.uk).

L. Ma is with the IBM T. J. Watson Research Center, Yorktown, NY 10598 USA (e-mail: maliang@us.ibm.com).

A. Swami is with the Army Research Laboratory, Adelphi, MD 20783 USA (e-mail: ananthram.swami.civ@mail.mil).

D. Towsley is with the University of Massachusetts Amherst, Amherst, MA 01003 USA (e-mail: towsley@cs.umass.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This consists of proofs and additional details.

Digital Object Identifier 10.1109/TNET.2016.2642185

or frequently occurring topologies in the past. Given such prediction capabilities, we are interested in two closely-related problems: (i) During network planning, how can we place monitors to ensure identifiability under predictable topology changes? (ii) At runtime, how can we adapt the monitor placement so as to maintain identifiability under unpredictable topology changes? In both problems, we wish to use as few monitors as possible to minimize cost.

A. Related Work

Based on how measurements are performed, existing solutions on monitor placement can be categorized as: (1) placement of monitors performing round-trip probing, (2) placement of monitors performing end-to-end probing.

In category (1), each monitor (aka beacon) independently computes metrics of a subset of links by sending round-trip probes to all possible destinations along its routing tree. The goal is thus to place a minimum number of monitors whose routing trees cover all the links. The problem is proved to be NP-hard, and algorithms based on set covering are proposed to select a sufficient set of monitors. Variations have also been proposed to cover all active links under a limited number of link failures or route changes [14], [15].

In category (2), monitors jointly infer link metrics from end-to-end measurements between themselves. Under the assumption that monitors can measure arbitrary cycles or paths (possibly) containing cycles, [9] derives the first necessary and sufficient condition on the network topology for identifying additive link metrics from end-to-end measurements. The condition is later modified by [10] to incorporate an additional constraint that measurement paths must be cycle-free. Based on the modified condition, [10] develops an algorithm to place a minimum number of monitors to identify *all* link metrics. Variations of the problem include [16], which combines the algorithm in [10] with a graph trimming algorithm to identify only the links of interest, and [17], which maximizes the number of identifiable links using a given number of monitors. All the above works assume a fixed network topology. Recently, attention is turned to support dynamic topology changes. Under the measurement model in [9], [18] studies a problem similar to [17] but wants to guarantee identifiability under up to k link failures. In this work, we adopt the measurement model in [10] but consider arbitrary topology changes.

As shown in [10], optimal monitor placement requires knowledge of the network structure in terms of *biconnected* and *triconnected* components (see Definition 2). For static graphs with m edges and n vertices, there are algorithms to compute biconnected components [19] and triconnected components, both in $O(m+n)$ time. For dense graphs, [21] proposes a sparsification technique that reduces the complexity to $O(n)$. For dynamic graphs, it is possible to reduce the complexity by reusing previous results. A dynamic-graph algorithm can be classified as either *partially* dynamic or *fully* dynamic, depending on whether it supports only one or both of edge insertion and deletion. Fully dynamic algorithms have been proposed in [22] to maintain biconnected components. In contrast, only partially dynamic algorithms exist for maintaining triconnected components after edge insertions [23], [24], and fully dynamic algorithms are known only in the special case of planar graphs [25]. In this work, we fill the gap by proposing an algorithm to update triconnected components in a general graph after edge deletion.

Key to robust monitor placement is the knowledge of potential network topologies after changes. Topology prediction has been studied in the context of wireless ad-hoc and vehicular networks, where techniques including adaptive filtering and fluid dynamic modeling have been proposed to predict link changes [11], [12] or extract topology snapshots from given mobility models [26]. In this work, we leverage such predictors and focus on developing robust monitor placement algorithms based on the predicted topologies.

B. Summary of Contributions

We study robust monitor placement for inferring additive link metrics from end-to-end measurements along cycle-free paths under dynamic topology changes. Our contributions are:

1) We develop robust monitor placement algorithms that place monitors to simultaneously identify a given set of topologies, including: (i) a *one-shot* placement algorithm that applies an existing algorithm designed for static networks to an aggregate topology, (ii) an *incremental* placement algorithm that sequentially places monitors in each topology, and (iii) an *optimal* placement algorithm that jointly considers all the topologies by casting the problem as a generalized hitting set problem. All the algorithms guarantee identifiability with different tradeoffs between the number of monitors and the computation complexity. We also provide an algorithm to identify and remove unnecessary monitors by solving a dual problem.

2) We show that the optimal monitor placement is NP-hard but can be approximated to a logarithmic factor by a greedy heuristic. We further identify several cases where the problem can be solved optimally in polynomial time. One case corresponds to static networks, explaining why the problem is solvable (by [10]) in this case.

3) We develop a dynamic graph decomposition algorithm that maintains the information needed by monitor placement (triconnected decomposition) for a dynamic graph undergoing edge deletions. This is the first dynamic triconnected decomposition algorithm that can handle edge deletion.

4) We extend the above solutions to address practical issues including unpredictable topology changes, bounded number of monitors, and arrival/departure of nodes.

5) We evaluate the proposed solutions on both random dynamic topologies and realistic dynamic topologies induced by mobility traces. Our results show that our dynamic graph decomposition algorithm can significantly improve the efficiency of the best existing algorithm, and our monitor placement algorithms can use a small percentage of monitors $(10-30\%)$ to maintain identifiability under hundreds of topology changes, while being highly robust to topology prediction errors.

The rest of the paper is organized as follows. Section II formulates the problem. Section III reviews existing results. Section IV presents new monitor placement algorithms, for

TABLE I
NOTATIONS

Symbol	Meaning
$\mathcal{G} = (V,\ L)$	graph with vertex set V and edge set L
$V_{\mathcal{G}'}$	set of vertices in graph $\mathcal{G}'$ (by default $V = V_{\mathcal{G}}$)
$L_{\mathcal{G}'}$	set of edges in graph $\mathcal{G}'$ (by default $L = L_{\mathcal{G}}$)
$l(v, u)$	edge connecting vertices v and u
$\mathcal{G} + l$ / $\mathcal{G} - l$	adding/deleting edge l from graph $\mathcal{G}$
$S_{\mathcal{G}'}$	separation vertices in graph $\mathcal{G}'$ (Section III-B)
$M_{\mathcal{G}'}$	monitors at non-separation vertices of graph $\mathcal{G}'$
$\mathcal{F}$	monitor selection constraints (Section IV-C)
$\mathcal{R}$	monitor removal constraints (Section IV-D)

which Section V identifies optimality conditions. Section VI presents a dynamic graph decomposition algorithm in support of monitor placement. Section VII discusses several extensions. Section VIII evaluates the proposed solutions. Section IX concludes the paper. Proofs are available in Appendix A of the supplementary file.

II. Problem Formulation

A. Notations

We will interchangeably use the terms "network (topology)" and "graph", "node" and "vertex", and "link" and "edge". Table I summarizes the main notations used in the paper.

B. Network Models

Consider a fixed set of nodes V which form a network with time-varying topologies. We assume that the network topologies are known and are represented by undirected graphs $\{\mathcal{G}_t : t = 1, \ldots, T\}$, where $\mathcal{G}_t = (V,\ L_t)$ is a graph representing the t-th topology.[1] For example, these topologies can be predicted using existing topology prediction models such as [11], [12]. We note that these topologies do not need to occur sequentially in time and do not need to cover all possible cases. Although we focus on link changes, node changes can also be handled by modeling them as special link changes; see Section VII-C.

C. Network Tomography

Given a topology $\mathcal{G} = (V,\ L)$ (omitting subscript t), network tomography aims at inferring the performance metrics of individual links in L from end-to-end metrics along a set of measurement paths P. In particular, we are interested in inferring *additive metrics* where the path metric equals the sum of the corresponding link metrics; examples of such metrics include delay, jitter, and log delivery rate (under the assumption of independent losses across links). A basic requirement of network tomography is the uniqueness of the solution, aka *identifiability*.

Definition 1: A network $\mathcal{G}$ is *identifiable* if all its link metrics can be uniquely determined from path metrics.

[1]Our monitor placement algorithms can handle arbitrary topologies, although their performance depends on the specific topologies; see Section V.

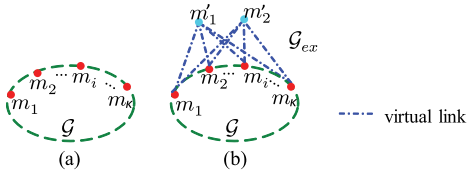


Fig. 1. (a) $\mathcal{G}$ with κ ($\kappa \geq 3$) monitors; (b) $\mathcal{G}_{ex}$ with two virtual monitors.

From linear algebra, we know that $\mathcal{G}$ is identifiable if and only if the rank of the measurement paths (when represented as vectors in the link space; see [10]) equals the number of links. We assume that we can only monitor paths that start/end at certain nodes designated as *monitors*. We further assume that monitors can control the routing of measurement packets as long as the path starts and ends at different monitors without incurring cycles, i.e., it is a *simple path* between monitors. In practice, we can set up such paths between monitors using technologies such as source routing, Multiprotocol Label Switching (MPLS), and Software-Defined Networking (SDN), and the cycle-free property guarantees that probes will not encounter forwarding loops. See discussions in [18] for more examples of such technologies. If $\mathcal{G}$ is identifiable under a given monitor placement, we say that this placement *identifies* $\mathcal{G}$.

D. Objective of Monitor Placement

Our main objective is to select a *smallest* subset of nodes as monitors to identify all the topologies of interest. We will discuss later (Section VII) how such a monitor placement can be augmented to maintain identifiability in case of unpredictable topology changes.

Remark: Note that our objective is to achieve link identifiability through monitor placement. While other objectives (e.g., load balancing) and design parameters (e.g., measurement paths and probing frequencies) are also important to the performance of network tomography, their optimization is beyond the scope of this paper and left to future work.

III. Monitor Placement for Static Networks

We start by reviewing existing results from [10] for a static network with a fixed topology. We then extract insights from these results to motivate our solutions for dynamic networks.

A. Identifiability Condition

Definition 1 does not allow efficient testing of identifiability as there are exponentially many measurement paths. Existing work [10] has established an equivalent condition in terms of an *extended graph* $\mathcal{G}_{ex}$ constructed as follows: given a network $\mathcal{G}$ with κ ($\kappa \geq 3$) monitors,[2] $\mathcal{G}_{ex}$ is obtained by adding two *virtual monitors* m'_1 and m'_2, and 2κ *virtual links* between each pair of virtual-actual monitors, as illustrated in Fig. 1. The identifiability of $\mathcal{G}$ is characterized by the following condition.

Theorem 1: [10] Given κ ($\kappa \geq 3$) monitors, $\mathcal{G}$ is identifiable *if and only if* its extended graph $\mathcal{G}_{ex}$ is 3-vertex-connected, i.e., it remains connected after removing any two nodes.

[2]It has been shown [10] that a network with more than one link needs at least three monitors to identify all link metrics.

B. Minimum Monitor Placement for Static Networks

Based on the identifiability condition in Theorem 1, [10] gives an algorithm, called *Minimum Monitor Placement (MMP)*, that places a minimum set of monitors to ensure identifiability. We briefly review MMP for completeness and refer to [10] for details. MMP works by decomposing $\mathcal{G}$ into subgraphs with certain properties defined as follows.

Definition 2: A *k-connected component* of $\mathcal{G}$ is a maximal sub-graph of $\mathcal{G}$ that is either (i) k-vertex-connected, or (ii) a clique with up to k vertices. The case of $k = 2$ is called a *biconnected component*, and $k = 3$ a *triconnected component*.

Biconnected components are subgraphs separated by cut-vertices,[3] and triconnected components are subgraphs separated by cut-vertices and 2-vertex cuts.[4] Common vertices between a given component and its neighboring components are called *separation vertices*.

In words, MMP works by ensuring that: (i) all nodes with degree (i.e., number of neighbors) 1 or 2 are monitors, (ii) all bi/tri-connected components with at least three nodes have at least three nodes that are separation vertices or monitors, and (iii) all connected components with at least three nodes have at least three monitors; see [10] for details.[5] In static networks, such a monitor placement is guaranteed to be sufficient and optimal in the following sense.

Theorem 2: [10] Given a fixed network topology $\mathcal{G}$, MMP places the minimum number of monitors to identify $\mathcal{G}$.

MMP can be implemented efficiently with a complexity of $O(|V| + |L|)$ for $\mathcal{G} = (V, L)$ [10]. The key step is to compute bi/tri-connected components of $\mathcal{G}$, which can be done in linear time using graph decomposition algorithms in [19] and [20].

Key observations: MMP places two types of monitors:

- deterministically placed monitors: nodes with degree less than three have to be monitors, as otherwise their neighboring links are not measurable by simple paths;
- randomly placed monitors: MMP ensures at least three nodes that are either separation vertices or monitors in each bi/tri/1-connected component, but the exact monitor locations can be arbitrary.

We will leverage these observations in deriving monitor placement algorithms for dynamic networks.

IV. Monitor Placement for Dynamic Networks

A dynamic network may have multiple topologies during its lifetime, represented by $\{\mathcal{G}_t : t = 1, \ldots, T\}$. Based on the identifiability condition in Theorem 1, the problem of robust monitor placement can be cast as follows: select a minimum set of nodes $M \subseteq V$ as monitors such that for each $t = 1, \ldots, T$, the extended graph $\mathcal{G}_{t,ex}$ constructed from topology $\mathcal{G}_t$ and monitors M is 3-vertex-connected.

Given the results for static networks (Section III), one can guarantee identifiability for all the T topologies by applying MMP to compute a monitor placement M_t for each $\mathcal{G}_t$, and then taking the union $M = \bigcup_{t=1}^{T} M_t$. Such a solution, however, may select redundant monitors. For example, to identify the topologies $\mathcal{G}_1$ and $\mathcal{G}_2$ in Fig. 2, MMP may select monitors $M_1 = \{a, b, c\}$ for $\mathcal{G}_1$, and $M_2 = \{c, f, h\}$ for $\mathcal{G}_2$, with a total of 5 monitors. However, by Theorem 1, M_2 already identifies both $\mathcal{G}_1$ and $\mathcal{G}_2$, i.e., two redundant monitors are selected by MMP. This example illustrates the need of new monitor placement algorithms that are specifically designed to handle multiple topologies. In this regard, we outline a set of solutions with different performance (wrt number of monitors) and complexity.

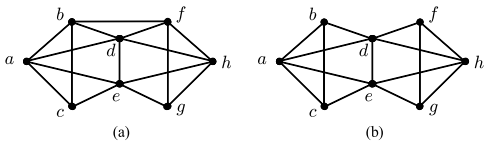


Fig. 2. Monitor placement for two topologies (the optimal monitor placement is $\{c, f, h\}$): (a) $\mathcal{G}_1$ (3-vertex-connected); (b) $\mathcal{G}_2$ (2-vertex-connected with a 2-vertex cut $\{d, e\}$).

A. One-Shot Placement

Intuitively, placing monitors to identify multiple topologies is at least as complex as placing monitors to identify a single topology. Interestingly, we show that this is achievable by leveraging a special property of identifiable networks. By Theorem 1, a network $\mathcal{G}$ that is identifiable under a given monitor placement M remains identifiable after adding links, because adding links maintains the 3-vertex-connectivity of $\mathcal{G}_{ex}$. Therefore, to identify topologies $\{\mathcal{G}_t = (V, L_t) : t = 1, \ldots, T\}$, it suffices for the monitor placement to identify their maximum common subgraph $\mathcal{G}_b := (V, \bigcap_{t=1}^{T} L_t)$, referred to as the *base graph*. Since each $\mathcal{G}_t$ can be generated from $\mathcal{G}_b$ by adding links, we have the following result.

Corollary 3: If a monitor placement M identifies the base graph $\mathcal{G}_b$ of $\{\mathcal{G}_t : t = 1, \ldots, T\}$, then it identifies each $\mathcal{G}_t$.

This result motivates a one-shot placement algorithm: first, compute the base graph $\mathcal{G}_b$ by identifying common links in all topologies, and then apply MMP to $\mathcal{G}_b$. Generally, one-shot placement uses more than the minimum number of monitors; however, it is very efficient and can be optimal in some cases (see Claim 12).

Complexity: The base graph can be computed in $O(T \min_t |L_t|)$ time, and applying MMP to the base graph takes $O(|V| + \min_t |L_t|)$ time. The overall complexity of one-shot placement is therefore $O(|V| + T \min_t |L_t|)$.

B. Incremental Placement

Alternatively, we can sequentially apply a variation of MMP to each topology, which takes into account the existing monitors to minimize the number of additional monitors.

The algorithm, named *Incremental Minimum Monitor Placement (IMMP)*, is presented in Algorithm 1. It differs from MMP [10] in that it takes an additional input of existing monitors M_0 and only returns the newly selected monitors M_a. Specifically, given a subgraph $\mathcal{D}$, let $V_{\mathcal{D}}$ denote all the nodes in $\mathcal{D}$, $S_{\mathcal{D}}$ the separation vertices in $\mathcal{D}$, and $M_{\mathcal{D}}$ the monitors at internal nodes (i.e., non-separation vertices) of $\mathcal{D}$. Note

[3] A vertex v is a cut-vertex of $\mathcal{G}$ if removing v increases the number of connected components in $\mathcal{G}$.

[4] A vertex pair $\{u, v\}$ is a 2-vertex cut of a biconnected component $\mathcal{B}$ if removing $\{u, v\}$ disconnects $\mathcal{B}$.

[5] Note that [10] assumes the network to be connected; in general, MMP should be applied to each connected component.

Algorithm 1 Incremental Minimum Monitor Placement (IMMP)

input : Current network topology $\mathcal{G}$ and existing monitors M_0
output: Newly placed monitors M_a s.t. $M_a \cup M_0$ identifies $\mathcal{G}$

1 $M_a \leftarrow \{\text{nodes with degree 1 or 2}\} \setminus M_0$;
2 **foreach** *connected component* $\mathcal{C}_k$ *of* $\mathcal{G}$ *with at least* 3 *nodes* **do**
3 partition $\mathcal{C}_k$ into biconnected components $\mathcal{B}_1, \mathcal{B}_2, \ldots$;
4 **foreach** *biconnected component* $\mathcal{B}_i$ *with at least* 3 *nodes* **do**
5 partition $\mathcal{B}_i$ into triconnected components $\mathcal{T}_1, \mathcal{T}_2, \ldots$;
6 **foreach** *triconnected component* $\mathcal{T}_j$ **do**
7 **if** $|S_{\mathcal{T}_j}| + |M_{\mathcal{T}_j}| < 3$ **then**
8 $M_a \leftarrow M_a \cup \{(3 - |S_{\mathcal{T}_j}| - |M_{\mathcal{T}_j}|)$ nodes randomly selected from $V_{\mathcal{T}_j} \setminus (S_{\mathcal{T}_j} \cup M_{\mathcal{T}_j})\}$;
9 **if** $|S_{\mathcal{B}_i}| + |M_{\mathcal{B}_i}| < 3$ **then**
10 $M_a \leftarrow M_a \cup \{(3 - |S_{\mathcal{B}_i}| - |M_{\mathcal{B}_i}|)$ nodes randomly selected from $V_{\mathcal{B}_i} \setminus (S_{\mathcal{B}_i} \cup M_{\mathcal{B}_i})\}$;
11 **if** $|M_{\mathcal{C}_k}| < 3$ **then**
12 $M_a \leftarrow M_a \cup \{(3 - |M_{\mathcal{C}_k}|)$ nodes randomly selected from $V_{\mathcal{C}_k} \setminus M_{\mathcal{C}_k}\}$;

that $M_{\mathcal{D}}$ includes both existing and newly placed monitors and can vary during monitor placement. IMMP first places monitors at non-monitor nodes with only one or two neighbors (line 1). It then decomposes the network into biconnected and triconnected components (lines 3 and 5), based on which it selects additional monitors if necessary such that each tri/bi/1-connected component with at least three nodes has at least three separation vertices/monitors (lines 8, 10, 12).

Based on the optimality of MMP (Theorem 2), we can show that IMMP is optimal in the following sense.

Corollary 4: Given a network topology $\mathcal{G}$ and existing monitors M_0, IMMP places a minimum set of additional monitors M_a such that $M_0 \cup M_a$ identifies $\mathcal{G}$.

The overall algorithm works as follows: for each $\mathcal{G}_t = \mathcal{G}_1, \ldots, \mathcal{G}_T$ ($M_0 = \emptyset$),

1) $M_{a,t} \leftarrow$IMMP$(\mathcal{G}_t,\ M_{t-1})$;
2) $M_t \leftarrow M_{t-1} \cup M_{a,t}$.

Then M_T is guaranteed to identify $\mathcal{G}_1, \ldots, \mathcal{G}_T$.

Complexity: Since IMMP has the same complexity as MMP, i.e., $O(|V| + |L_t|)$ for each $\mathcal{G}_t$ [10], the overall complexity of the incremental placement algorithm is $O(T|V| + \sum_{t=1}^{T} |L_t|)$. Note that this analysis assumes that IMMP (lines 3 and 5) uses the algorithms in [19] and [20] to compute the bi/tri-connected components from scratch for each $\mathcal{G}_t$. We will later discuss dynamic graph decomposition algorithms that can reduce the complexity by reusing previous results if $\mathcal{G}_t$ is structurally similar to a previously decomposed topology (see Section VI).

Algorithm 2 Feasible Monitor Placement (FMP)

input : Network topology $\mathcal{G}$
output: Monitor placement constraints $\mathcal{F}$

1 **foreach** *node* v *with degree* 1 *or* 2 **do**
2 add $(\{v\},\ 1)$ to $\mathcal{F}$;
3 **foreach** *connected component* $\mathcal{C}_k$ *of* $\mathcal{G}$ *with at least* 3 *nodes* **do**
4 partition $\mathcal{C}_k$ into biconnected components $\mathcal{B}_1, \mathcal{B}_2, \ldots$;
5 **foreach** *biconnected component* $\mathcal{B}_i$ *with at least* 3 *nodes* **do**
6 partition $\mathcal{B}_i$ into triconnected components $\mathcal{T}_1, \mathcal{T}_2, \ldots$;
7 **foreach** *triconnected component* $\mathcal{T}_j$ **do**
8 **if** $|S_{\mathcal{T}_j}| < 3$ **then**
9 add $(V_{\mathcal{T}_j} \setminus S_{\mathcal{T}_j},\ 3 - |S_{\mathcal{T}_j}|)$ to $\mathcal{F}$;
10 **if** $|S_{\mathcal{B}_i}| < 3$ **then**
11 add $(V_{\mathcal{B}_i} \setminus S_{\mathcal{B}_i},\ 3 - |S_{\mathcal{B}_i}|)$ to $\mathcal{F}$;
12 add $(V_{\mathcal{C}_k},\ 3)$ to $\mathcal{F}$;

C. Joint Placement

Despite being locally optimal (Corollary 4), the overall placement by the incremental algorithm is generally suboptimal due to the lack of a *joint* consideration of all topologies. Jointly considering all topologies is highly nontrivial, as each topology can have exponentially many, equally optimal monitor placements, corresponding to all combinations of possible monitor locations within each bi/tri-connected component. The brute-force strategy of enumerating all possible placements for individual topologies to find a minimum union is clearly inefficient. Our idea in addressing this issue is to decouple the problem into two stages: (i) constraint characterization and (ii) monitor selection.

1) Constraints on Monitor Placement: Our key insight is that all possible placements generated by MMP can be succinctly encoded into a set of constraints. Specifically, instead of randomly picking one placement as in MMP, we record the constraints on monitor placement in the form of set-integer pairs $\mathcal{F} = \{(S_i,\ k_i) :\ i = 1, 2, \ldots\}$, where each $S_i \subseteq V$ is a set of candidate monitors, and k_i is the minimum number of monitors selected from this set. For example, if MMP places a monitor at a randomly selected node in S, we can represent all possible placements by a constraint $(S,\ 1)$.

Given a topology $\mathcal{G}$, these constraints can be computed efficiently using an algorithm called *Feasible Monitor Placement (FMP)*, shown in Algorithm 2. FMP follows a procedure similar to MMP, except that instead of selecting specific nodes as monitors, it records the constraints on where monitors should be placed such that $\mathcal{G}$ is identifiable. Specifically, let $V_{\mathcal{D}}$ and $S_{\mathcal{D}}$ be defined as in Algorithm 1. FMP generates one constraint for each node with only one or two neighbors (line 2) to indicate that such nodes must be monitors. It then decompose the network into biconnected and triconnected components (lines 4, 6), based on which it generates one constraint for each tri/bi/1-connected component with at least three nodes to ensure that it has at least three nodes that are separation vertices or monitors (lines 9, 11, 12).

We can show that the resulting constraints are both sufficient and necessary for identifying $\mathcal{G}$.

Lemma 5: Any monitor placement $M \subseteq V$ identifies $\mathcal{G}$ if and only if M satisfies the constraints $\mathcal{F}$ computed by FMP, i.e., $|M \cap S_i| \geq k_i$ for all $(S_i,\ k_i) \in \mathcal{F}$.

Discussion: It is possible that not all constraints are needed, e.g., if the total number of monitors required by the triconnected components within a biconnected component exceeds the number of monitors required by the biconnected component, then we do not need a separate constraint for this biconnected component. We can avoid redundant constraints by counting the number of monitors in each bi/tri/1-connected component according to existing constraints (by mimicking MMP) and adding a new constraint to $\mathcal{F}$ only if the current number of monitors is not sufficient.

2) Constrained Monitor Selection: Given the constraints $\mathcal{F}$ obtained by applying FMP to each topology $\mathcal{G}_t$ $(t = 1, \ldots, T)$, the monitor placement problem is converted into a problem of selecting a minimum subset $M \subseteq V$ such that $|M \cap S_i| \geq k_i$ for all $(S_i,\ k_i) \in \mathcal{F}$. We refer to this problem as the *minimum hitting set problem (min-HSP)* with input $(V,\ \mathcal{F})$. In the special case of $k_i \equiv 1$, it becomes the classic hitting set problem (HSP). Because the constraints computed by FMP are necessary and sufficient for achieving identifiability (Lemma 5), we can obtain an optimal monitor placement by solving the corresponding min-HSP optimally.

Theorem 6: Let $\mathcal{F}$ be the overall set of constraints computed by applying FMP to each topology $\mathcal{G}_t$ $(t = 1, \ldots, T)$. Then the optimal solution to min-HSP$(V, \mathcal{F})$ yields an optimal (i.e., minimum) monitor placement for identifying $\mathcal{G}_1, \ldots, \mathcal{G}_T$.

The challenge is that min-HSP is NP-hard since HSP is NP-hard. Although given constraints $\mathcal{F}$, the problem of constrained monitor selection is a special case of min-HSP, we show that it is still NP-hard by a reduction from HSP.

Theorem 7: The optimal monitor placement for arbitrary topologies $\mathcal{G}_t$ $(t = 1, \ldots, T)$ is NP-hard.

Greedy approximation: In our problem, we can speed up computation by first filtering out nodes that must be monitors (i.e., nodes with degree less than three in at least one topology) and then applying the greedy heuristic. For our problem, the greedy heuristic works as follows: while there are unsatisfied constraints, select the monitor that helps in satisfying the maximum number of unsatisfied constraints, i.e., given the current monitors M, select v as the next monitor such that v is in the maximum number of sets among $\{S_i : (S_i,\ k_i) \in \mathcal{F}, |S_i \cap M| < k_i\}$. It can be shown that the greedy heuristic achieves a logarithmic approximation as follows.

Theorem 8: The greedy heuristic for min-HSP$(V,\ \mathcal{F})$ achieves an approximation ratio of $(1 + \log|\mathcal{F}|)$, i.e., the number of monitors selected by the greedy heuristic is at most $(1 + \log|\mathcal{F}|)$ times larger than the minimum number of monitors.

Complexity: Together, FMP and the greedy heuristic for min-HSP provide a joint monitor placement algorithm. FMP has the same complexity as MMP, which is $O(|V| + |L_t|)$ for each $\mathcal{G}_t$ $(t = 1, \ldots, T)$ [10] if employing the static-graph algorithms in [19] and [20] to compute the bi/tri-connected components. As in IMMP, we can use dynamic-graph algorithms discussed in Section VI to speed up computation for $t > 1$. The greedy heuristic for min-HSP has complexity $O(T|V|^2)$, as there are $O(T|V|)$ constraints and an $O(|V|)$-complexity update upon satisfying each constraint (to compute the number of unsatisfied constraints each candidate monitor is involved in). Thus, the overall complexity is $O(T|V|^2)$.

Algorithm 3 Redundant Monitor Discovery (RMD)

```
   input : Network topology G, monitors M that identify G
   output: Monitor removal constraints R
1  foreach node v with degree 1 or 2 do
2  |  add ({v}, 0) to R;
3  foreach connected component C_k of G with at least 3
   nodes do
4  |  partition C_k into biconnected components B_1, B_2, ...;
5  |  foreach biconnected component B_i with at least 3
   |  nodes do
6  |  |  partition B_i into triconnected components
   |  |  T_1, T_2, ...;
7  |  |  foreach triconnected component T_j do
8  |  |  |  if |S_{T_j}| < 3 then
9  |  |  |  |  add (M_{T_j}, |M_{T_j}| - 3 + |S_{T_j}|) to R;
10 |  |  if |S_{B_i}| < 3 then
11 |  |  |  add (M_{B_i}, |M_{B_i}| - 3 + |S_{B_i}|) to R;
12 |  add (M_{C_k}, |M_{C_k}| - 3) to R;
```

D. Refinement of Placement

Due to the hardness of the optimal solution, the computed monitor placement generally contains more than the minimum number of monitors. A natural question is therefore how to refine this placement by removing redundant monitors without losing identifiability. As shown below, the problem of (redundant) monitor removal can also be decoupled into two stages.

1) Constraints on Monitor Removal: The problem of characterizing constraints on monitor removal is similar to that of characterizing constraints on monitor selection (Section IV-C.1), and thus can be solved by following similar steps.

The algorithm, referred to as *Redundant Monitor Discovery (RMD)*, is summarized in Algorithm 3. Define $S_{\mathcal{D}}$ and $M_{\mathcal{D}}$ as in Algorithm 1. Note that in contrast to Algorithm 1 where $M_{\mathcal{D}}$ varies during monitor placement, here $M_{\mathcal{D}}$ is based on the given monitor placement and thus fixed. RMD is analogous to FMP, except that it computes constraints for removing monitors (lines 2, 9, 11, 12). Each constraint is also represented by a set-integer pair $(S_i,\ k_i)$, which means that no more than k_i monitors from S_i can be removed.

It is easy to verify that when removing monitors from a monitor placement M that identifies $\mathcal{G}$, the remaining monitors satisfy the constraints computed by FMP if and only if the removed monitors satisfy the constraints computed by RMD. This duality immediately yields the following result.

Lemma 9: Given a monitor placement M that identifies $\mathcal{G}$, $M \setminus M'$ identifies $\mathcal{G}$ for any $M' \subset M$ if and only if M' satisfies $\mathcal{R}$ computed by RMD, i.e., $|M' \cap S_i| \leq k_i$ for all $(S_i,\ k_i) \in \mathcal{R}$.

Discussion: As in FMP, it is also possible for RMD to generate redundant constraints. We can avoid this by removing redundant constraints before adding each new constraint, i.e., when adding a constraint $(S,\ k)$, all existing constraints $(S',\ k')$ with $S' \subseteq S$ and $k' \geq k$ can be removed.

2) Constrained Monitor Removal: Given an initial placement M_0 that identifies all $\mathcal{G}_t$ $(t = 1, \ldots, T)$ and the constraints $\mathcal{R}$ computed by applying RMD to each $\mathcal{G}_t$ and M_0, the problem of removing redundant monitors in M_0 becomes a problem of selecting a maximum subset $M' \subset M_0$ such that $|M' \cap S_i| \leq k_i$ for all $(S_i,\ k_i) \in \mathcal{R}$. We refer to this problem as the *maximum hitting set problem (max-HSP)* with input $(M_0, \mathcal{R})$. The duality between the minimum monitor selection and the maximum redundant monitor removal implies an alternative way of computing an optimal placement as follows.

Corollary 10: Let $\mathcal{R}$ be the overall set of constraints computed by applying RMD to each of $\mathcal{G}_1, \ldots, \mathcal{G}_T$ with an initial placement $M_0 = V$, and M' be the optimal solution to max-HSP($V,\ \mathcal{R}$). Then $V \setminus M'$ is an optimal monitor placement for identifying $\mathcal{G}_1, \ldots, \mathcal{G}_T$.

Unfortunately, max-HSP is again NP-hard. In fact, it is exactly as hard as min-HSP because solving a min-HSP for input $(V,\ \{(S_i,\ k_i) :\ i = 1, 2, \ldots\})$ is equivalent to solving a max-HSP for input $(V,\ \{(S_i,\ |S_i| - k_i) :\ i = 1, 2, \ldots\})$.

Greedy approximation: We can apply a greedy heuristic similar to the one used in Section IV-C.2: while there are redundant monitors (i.e., monitors such that removing any one of them does not violate any constraint), remove the redundant monitor that is involved in the minimum number of constraints. We bound the performance of this greedy heuristic as follows.

Theorem 11: The greedy heuristic for max-HSP($V,\ \mathcal{R}$) achieves an approximation ratio of $1/|\mathcal{R}|$, i.e., the number of redundant monitors selected by the greedy heuristic is at most $|\mathcal{R}|$ times smaller than the maximum number of redundant monitors.

Complexity: Although max-HSP has the same complexity as min-HSP on the same input, the actual complexity of monitor removal depends on the size of the initial placement $|M_0|$, which can be much smaller than $|V|$. Specifically, RMD has the same complexity as FMP, i.e., $O(T|V| + \sum_t |L_t|)$ for processing $\mathcal{G}_1, \ldots, \mathcal{G}_T$. The greedy heuristic for max-HSP has complexity $O(T|V| \cdot |M_0|)$ to select from a size-$|M_0|$ set under $O(T|V|)$ constraints. The overall complexity of RMD and greedy max-HSP is therefore $O(T|V| \cdot |M_0| + \sum_t |L_t|)$.

Example: Consider a network with the topologies $\mathcal{G}_1$ and $\mathcal{G}_2$ in Fig. 3. The one-shot placement first obtains the base graph $\mathcal{G}_b$ and then applies MMP to $\mathcal{G}_b$, which may select monitors $M^{\text{one}} = \{a, d, f, h, g\}$. Note that MMP is a random algorithm; in this case, it may select c instead of d. The incremental placement first applies MMP to $\mathcal{G}_1$, which may select $\{b, c, f, h\}$ as monitors, and then applies IMMP to $\mathcal{G}_2$, which may select additional monitors $\{a, g\}$, yielding $M^{\text{inc}} = \{b, c, f, h, a, g\}$. The joint placement first computes the constraints: $\mathcal{F}_1 = \{(\{a, b\}, 1), (\{a, b, c, d\}, 2), (\{h\}, 1), (\{f, h, g\}, 2)\}$ for $\mathcal{G}_1$, and $\mathcal{F}_2 = \{(\{a\}, 1), (\{g\}, 1), (\{c, d, e, f, g, h\}, 2)\}$ for $\mathcal{G}_2$. It then uses the greedy heuristic to solve min-HSP for input $(V,\ \mathcal{F}_1 \cup \mathcal{F}_2)$, which yields $M^{\text{join}} = \{a, g, h, b\}$ (b may be replaced by c or d). The refined placement based on initial placement V generates the same result. Since $\mathcal{G}_1$ already requires four monitors, both the joint and the refined placements are optimal in this example.

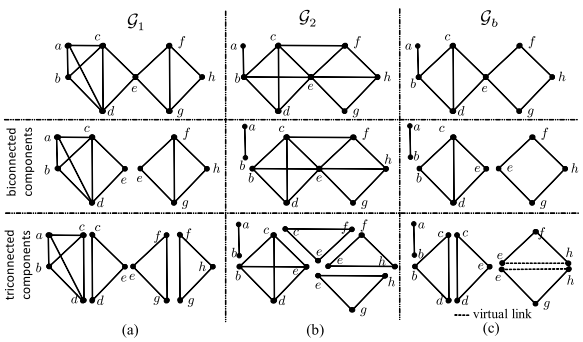


Fig. 3. Example of placement algorithms (an optimal monitor placement is $\{a, d, g, h\}$): (a) $\mathcal{G}_1$ and its decomposition; (b) $\mathcal{G}_2$ and its decomposition; (c) base graph $\mathcal{G}_b$ of $\mathcal{G}_1$ and $\mathcal{G}_2$ and the decomposition of $\mathcal{G}_b$.

V. Optimality Conditions

We have seen from Theorem 7 that, unlike the monitor placement problem in a static network, optimal monitor placement in a dynamic network is generally hard to compute. This motivates us to identify conditions under which the problem can be solved optimally.

A. Optimality Condition for One-Shot Placement

A lower bound on the number of monitors needed by a robust placement is the maximum number of monitors placed by MMP in any one of the topologies. Therefore, if the number of monitors needed to identify the base graph $\mathcal{G}_b$ matches the lower bound, the one-shot placement is optimal.

Claim 12: If $\exists t \in \{1, \ldots, T\}$ such that the number of monitors placed by MMP in $\mathcal{G}_t$ equals the number of monitors placed by MMP in $\mathcal{G}_b$, then the one-shot placement is optimal.

Remark: Intuitively, this condition is satisfied when the link sets of different topologies have a (roughly) nested structure. A special case of this condition is when $\mathcal{G}_b = \mathcal{G}_t$ for some t, i.e., $\mathcal{G}_t$ is a subgraph of all the other topologies $\mathcal{G}_{t'}$ for $t' \in \{1, \ldots, T\} \setminus t$.

B. Optimality Condition for Incremental Placement

Generally, the performance of incremental placement is sensitive to the order of applying IMMP to different topologies. In a special case where all topologies are 3-vertex-connected, however, the order no longer matters, because there is only one triconnected component in the network, and we know by MMP [10] that we only need three monitors in each $\mathcal{G}_t$, arbitrarily placed. This automatically implies the following.

Claim 13: If $\mathcal{G}_1, \ldots, \mathcal{G}_T$ are all 3-vertex-connected, then incremental placement is optimal regardless of the order of processing the topologies.

Remark: Although under this condition, incremental placement degenerates into MMP (a monitor placement that identifies any $\mathcal{G}_t$ identifies all the other topologies), it is different from the one-shot placement. In particular, the base graph of multiple 3-vertex-connected graphs may not be 3-vertex-connected, and thus the one-shot placement may need more than three monitors.

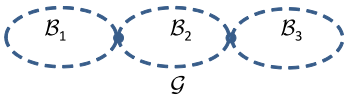


Fig. 4. A network with three biconnected (and also triconnected) components $\mathcal{B}_1, \ldots, \mathcal{B}_3$.

C. Optimality Condition for Joint Placement

As explained in Section IV-C.2, the difficulty in solving the monitor placement problem optimally is caused by the hardness of min-HSP. Therefore, any condition that allows min-HSP to be solved optimally (in polynomial time) is an optimality condition for a (polynomial-time) joint placement algorithm. To this end, we establish a condition for solving min-HSP that generalizes a well-known condition for solving HSP.

1) Equivalent Formulation: It is well known that HSP is equivalent to the *set cover problem (SCP)*. Naturally, min-HSP can also be represented by a variation of SCP, known as the *set multi-cover problem (SMCP)* [27]. Given an input $(U, \mathcal{E})$ for min-HSP, where $\mathcal{E} = \{(S_i, k_i) : S_i \subseteq U\}$ means that at least k_i items must be selected from S_i, we can construct an input $(\tilde{U}, \tilde{\mathcal{E}})$ for SMCP, where $\tilde{U} = \mathcal{E}$ specifies each "item" S_i and its minimum frequency of coverage k_i, and $\tilde{\mathcal{E}} = \{\mathcal{S}_e : e \in U\}$ specifies the "sets" used to cover S_i's ($\mathcal{S}_e := \{S_i : e \in S_i\}$). The min-HSP is equivalent to SMCP that selects the minimum number of sets $\mathcal{E}' \subseteq \tilde{\mathcal{E}}$ to cover each S_i at least k_i times. We will work on SMCP for ease of presentation.

SMCP is NP-hard and can only be approximated within a factor of $(1 + \max_{e \in U} \log |\mathcal{S}_e|)$ by the greedy heuristic [28]. For inputs with certain properties as shown below, however, SMCP can be solved optimally in polynomial time.

2) Existing Optimality Condition: It is known that SCP is polynomial-time solvable when its input satisfies a condition known as the *consecutive ones property (C1P)* [29]. In words, a SCP has C1P if there is a permutation of the items such that each set covers a set of consecutive items. It can be shown that SMCP is also polynomial-time solvable under C1P. Specifically, under C1P, the constraint matrix for the integer linear programming (ILP) representation of SMCP is totally unimodular [29], and thus the linear programming (LP) relaxation gives an integral solution which is optimal.

This condition is, however, too strong. For example, even the SMCP corresponding to placing monitors in a single topology can violate C1P. Consider the network in Fig. 4, where the monitor placement constraints computed by FMP are $\{(B_1, 2), (B_2, 1), (B_3, 2), (V, 3)\}$ (B_i denotes the set of internal nodes in component $\mathcal{B}_i$ for $i = 1, \ldots, 3$). Since $\{B_1, V\}$, $\{B_2, V\}$, and $\{B_3, V\}$ all contain common nodes, there is no permutation of $\{B_1, B_2, B_3, V\}$ such that the sets containing a given node are always consecutive, i.e., the corresponding SMCP violates C1P. However, we know that the optimal monitor placement in this case is polynomial-time solvable (by MMP).

3) Generalized Optimality Condition: Below, we give a more general condition for solving SMCP. Our condition is motivated by the following observation: for an item belonging to nested sets, i.e., each set is a subset of another, covering it by the set with the largest cardinality is always optimal as this set maximizes the coverage of other items. This obsrevation inspires the following condition.

Theorem 14: If the items can be represented by nodes in a rooted tree such that each set covers a set of consecutive nodes along a leaf-to-root path in the tree, then SMCP (and the corresponding min-HSP) is polynomial-time solvable.

We prove this result by constructing an algorithm that solves SMCP optimally under the above condition. Let $\mathcal{I} = \{(e, k_e) : e \in U\}$ denote the items and their required frequency of coverage, and $\mathcal{E} = \{S_i : S_i \subseteq U\}$ the sets. The algorithm, referred to as *Leaf-based Greedy Cover (LGC)*, works as follows: while $\exists$ a non-sufficiently covered item (i.e., e such that the number of selected sets covering e is less than k_e),

1) for an arbitrary, non-sufficiently covered leaf item[6] e, select the set S that covers the most non-sufficiently covered items among sets covering e;
2) update the remaining frequency of coverage required by each item and the remaining sets.

The idea of the proof is to show that under the condition in Theorem 14, LGC is optimal; see the proof in Appendix A.

Translated back to the joint placement problem, the condition in Theorem 14 requires that the sets of candidate monitors computed by FMP be representable by nodes in a rooted tree such that only consecutive sets on the same leaf-to-root path may overlap. One sufficient condition is as follows.

Corollary 15: Let $\mathcal{F} = \{(S_i, k_i), i = 1, 2, \ldots\}$ be the constraints computed by FMP for $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$. If for any $i \neq j$, S_i and S_j are either disjoint or nested (one is a subset of the other), then the optimal monitor placement for $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ can be computed in polynomial time.

Alternatively, the result of Corollary 15 can be proved using the theory of minimizing a monotone concave cost function under laminar covering constraints [30]. Specifically, our cost function (number of monitors) is linear, and our constraints S_i's form a *laminar family* under the condition in Corollary 15. Thus, we can apply the algorithm in [30] (for F_3) to select monitors optimally in $O(|V| \log^2 |V|)$ time.

Remark: A special case satisfying the condition in Corollary 15 is that of placing monitors in a single topology, where S_i's are internal nodes in tri/bi/1-connected components of the topology. In this sense, Corollary 15 provides an explanation on why computing the optimal monitor placement for a single topology is easy (solved by MMP) but computing that for multiple topologies is generally hard. In general, this condition holds if there is only splitting or merging of components during topology changes. For example, in Fig. 2, $\mathcal{G}_1$ contains a single triconnected component that is split into two in $\mathcal{G}_2$, and the monitor placement constraint is $\mathcal{F} = \{(\{a, b, c\}, 1), (\{f, g, h\}, 1), (\{a, b, c, d, e, f, g, h\}, 3)\}$, which satisfies the condition in Corollary 15.

VI. Efficient Implementation by Dynamic Graph Decomposition

At the core of the monitor placement algorithms is the computation of bi/tri-connected components in dynamic graphs.

[6]Precisely, e is such that e is not sufficiently covered, but all items below e in the rooted tree are sufficiently covered.

Although one can apply the graph decomposition algorithms developed for static graphs in [19] and [20] to compute these components from scratch for each topology, such an approach may incur redundant computation if some topologies, especially the consecutive ones, share similar structures. In this section, we investigate the use of dynamic graph algorithms to speed up the computation by reusing previous decomposition results. Here we use "vertex/edge" instead of "node/link" in the convention of graph algorithms.

A. Existing Solutions

For biconnected decomposition, [22] has proposed a *fully* dynamic algorithm to maintain biconnected components upon edge insertion/deletion in $O(\log^5 |V|)$ amortized time per update. For triconnected decomposition, however, only *partially* dynamic algorithms exist. Specifically, [23] proposes an algorithm based on a data structure called SPQR-tree to maintain the triconnected components upon edge/node insertion in $O(|V|log|V| + k)$ time per k updates. A similar algorithm based on a data structure called cycle tree is proposed in [24], which has a complexity of $O(|V| + k\alpha(k, |V|))$ for inserting k edges into an empty graph of $|V|$ vertices ($\alpha(|L|, |V|)$ is the inverse Ackermann function).

What is missing is an algorithm to update triconnected decomposition under edge deletion (which also handles the case of node deletion as discussed in Section VII-C). In the sequel, we present such an algorithm, which together with the algorithms handling edge insertion in [23], [24] forms a first *fully* dynamic algorithm for triconnected decomposition. As in [23], [24], we focus on the deletion of a single edge, which can be repeated to handle the deletion of multiple edges.

B. Unique Representation of Triconnected Decomposition

Although triconnected components are separated from each other by cut-vertices and 2-vertex cuts, not all subgraphs separated by these satisfy Definition 2 for $k = 3$. To fix this issue, an iterative procedure is proposed in [10] to add *virtual edges* between vertices in 2-vertex cuts, which results in a *non-unique* set of triconnected components that depend on the order of adding virtual edges. We avoid such ambiguity by introducing another type of component called *polygon*.

Definition 3: Given a triconnected decomposition of $\mathcal{G}$,

- two triangle components are *combinable* if they share a virtual edge and no other component shares this edge;
- after combining all combinable triangles (by removing the shared virtual edges), each component formed by one or more triangles is a *polygon*.

For example, subgraph $\mathcal{D}$ in Fig. 8 (a) is a polygon formed by combining four triangles. Note that a triangle is also considered a polygon. Since polygon vertices are either degree-2 vertices (which must be monitors) or separation vertices (which do not need to be monitors), for the application of monitor placement, it suffices to find the polygons and the triconnected components that are not triangles. In the sequel, we simply refer to triconnected components that are not triangles as "triconnected components". It has been shown in Lemma 2 of [20] that the decomposition of a graph into such triconnected components and polygons is unique.[7]

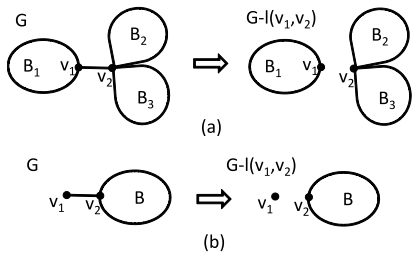


Fig. 5. Remove an edge from a bond: (a) not generating isolated vertex, (b) generating isolated vertex.

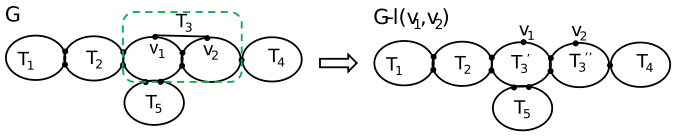


Fig. 6. Remove an edge from a triconnected component with more than three vertices.

C. Observations

We start with several observations that guide our algorithm design. The correctness of these observations will be justified later. Suppose that edge $l(v_1, v_2)$ between vertices v_1 and v_2 is deleted. Then depending on which components $l(v_1, v_2)$ belongs to, we have the following observations.

Observation A: If $l(v_1, v_2)$ is in a bond (see Fig. 5), then the connected component containing the edge is split into two connected components and the decomposition within each new connected component remains the same. Note that deleting $l(v_1, v_2)$ may create isolated vertices (e.g., v_1 in Fig. 5 (b)) which form trivial triconnected components.

Observation B: If $l(v_1, v_2)$ is in a triconnected component $\mathcal{T}$ with more than three vertices, then the deletion of $l(v_1, v_2)$ will not cause any change if it is on the boundary of $\mathcal{T}$ (i.e., $\{v_1, v_2\}$ forms a 2-vertex cut), but may cause $\mathcal{T}$ to be split into multiple triconnected components/polygons if it is in the interior of $\mathcal{T}$ (see Fig. 6). Meanwhile, decomposition outside $\mathcal{T}$ remains the same.

Observation C: If neither Observation A nor Observation B applies, then $l(v_1, v_2)$ must reside in one or more polygons. We have the following subcases:

1) If $l(v_1, v_2)$ appears in three or more polygons, then the triconnected decomposition remains the same;
2) If $l(v_1, v_2)$ appears in two and only two polygons, then the polygons are combined into a new polygon (see Fig. 7);
3) If $l(v_1, v_2)$ is in one and only one polygon $\mathcal{D}$ (see Fig. 8), then the removal of $l(v_1, v_2)$ destroys $\mathcal{D}$. If $\mathcal{D}$ contains an edge $l' \neq l(v_1, v_2)$ that is not shared with any other component, then l' forms a new triconnected component (a bond); moreover, if $\mathcal{D}$ contains a virtual edge that is no longer needed (e.g., $l(w_3, w_4)$ in Fig. 8 (a)), then this virtual edge also needs to be removed, which may cause updates to the components containing the edge

[7]Note that polygons are considered a type of "triconnected components" in [20].

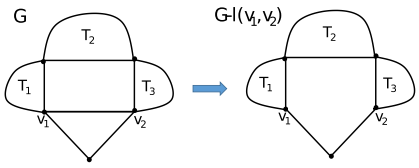


Fig. 7. Remove an edge belonging to only two polygons.

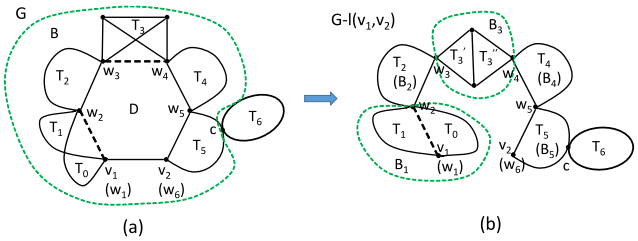


Fig. 8. Remove an edge belonging to only one polygon. (a) Before removing $l(v_1, v_2)$. (b) After removing $l(v_1, v_2)$.

according to Observations B-C (1-2) (e.g., splitting $\mathcal{T}_3$ in Fig. 8 (a) into $\mathcal{T}_3'$ and $\mathcal{T}_3''$ in Fig. 8 (b)).

Observations A-C cover all possible cases of deleting a single edge. In the sequel, we will justify these observations and develop efficient algorithms to update the triconnected components/polygons according to these observations.

D. Triconnected Decomposition After Edge Deletion

Let Φ denote the set of triconnected components/polygons of $\mathcal{G}$. In implementation, it suffices to record the internal and the separation vertices in each component, which together with the adjacency matrix of $\mathcal{G}$ specify the structure and the inter-connectivity of all components. We propose Algorithm 4, *Triconnected decomposition after Edge Deletion (TED)*, to update Φ when a single edge $l(v_1, v_2)$ is removed. Depending on the location of the edge, there are three cases, each corresponding to one observation in Section VI-C:

(1) $l(v_1, v_2)$ *is in a bond* (Observation A). Since a bond is a degenerate triconnected component, deleting $l(v_1, v_2)$ deletes the entire component as in line 5. Moreover, if v_i $(i = 1, 2)$ becomes isolated, then v_i forms a new degenerate triconnected component representing a single vertex, handled by line 3.

(2) $l(v_1, v_2)$ *is in a triconnected component* $\mathcal{T}$ *with more than three vertices* (Observation B). If $\{v_1, v_2\}$ is a 2-vertex cut, then the deletion of $l(v_1, v_2)$ has no effect on the triconnected decomposition as it will be replaced by a virtual edge, and hence TED ignores this case. Otherwise, as illustrated by Fig. 6, $\mathcal{T}$ may be split into multiple triconnected components/polygons. To update the triconnected decomposition, we first apply the algorithm in [20] to compute the triconnected decomposition of $\mathcal{T} - l(v_1, v_2)$, and then replace $\mathcal{T}$ by the triconnected components/polygons of $\mathcal{T} - l(v_1, v_2)$ (line 7). Furthermore, the decomposition of $\mathcal{T} - l(v_1, v_2)$ may create new polygons. If such a polygon shares a virtual edge with one and only one other polygon (outside $\mathcal{T}$), then according to Definition 3, we need to remove the virtual edge and combine the polygons. This is handled by calling an auxiliary algorithm, *Combine Polygon after Edge Deletion (CPED)*, presented in Algorithm 5 (line 8).

Algorithm 4 Triconnected decomposition after Edge Deletion (TED)

input : Triconnected decomposition Φ of $\mathcal{G}$ and deleted edge $l(v_1, v_2)$
output: Triconnected decomposition Φ' of $\mathcal{G} - l(v_1, v_2)$

1 **if** $l(v_1, v_2)$ *is in a bond* $\mathcal{B}$ **then**
2 **if** v_i $(i = 1, 2)$ *is isolated in* $\mathcal{G} - l(v_1, v_2)$ **then**
3 $\Phi \leftarrow$ replace $\mathcal{B}$ by a degenerate component representing v_i;
4 **else**
5 $\Phi \leftarrow$ delete $\mathcal{B}$;
6 **else if** $l(v_1, v_2)$ *is in a triconnected component* $\mathcal{T}$ *with more than three vertices* ***and*** $\{v_1, v_2\}$ *is not a 2-vertex cut* **then**
7 $\Phi \leftarrow$ replace $\mathcal{T}$ by triconnected components/polygons of $\mathcal{T} - l(v_1, v_2)$;
8 $\Phi \leftarrow$ CPED(Φ, {2-vertex cuts of $\mathcal{T}$});
9 **else if** $l(v_1, v_2)$ *is in two and only two polygons* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ **then**
10 $\Phi \leftarrow$ combine $\mathcal{D}_1$ and $\mathcal{D}_2$ into a new polygon;
11 **else if** $l(v_1, v_2)$ *is in one and only one polygon* $\mathcal{D}$ *with vertices* $w_1, \ldots, w_\mu$ $(w_1 = v_1,\ w_\mu = v_2)$ **then**
12 $\Phi \leftarrow$ delete $\mathcal{D}$;
13 **foreach** $i = 1, 2, \ldots, \mu - 1$ **do**
14 **if** $\{w_i, w_{i+1}\}$ *is a 2-vertex cut* **then**
15 **if** $\{w_i, w_{i+1}\}$ *is only shared by* $\mathcal{D}$ *and a triconnected component* $\mathcal{T}_i$ ***and*** $l(w_i, w_{i+1})$ *is a virtual edge* **then**
16 $\Phi \leftarrow$ replace $\mathcal{T}_i$ by triconnected components/polygons of $\mathcal{T}_i - l(w_i, w_{i+1})$;
17 $\Phi \leftarrow$ CPED(Φ, {2-vertex cuts of $\mathcal{T}_i$});
18 **else**
19 $\Phi \leftarrow$ CPED(Φ, $\{w_i, w_{i+1}\}$);
20 **else**
21 $\Phi \leftarrow$ create a new triconnected component for the bond formed by $l(w_i, w_{i+1})$;
22 return $\Phi' \leftarrow$ updated Φ;

(3) $l(v_1, v_2)$ *is in one or more polygons* (Observation C). There are three subcases: (i) $l(v_1, v_2)$ resides in three or more polygons (Observation C-1). In this case, even if $l(v_1, v_2)$ is removed, v_1 and v_2 are still connected by a virtual edge, and hence the triconnected decomposition remains unchanged. (ii) $l(v_1, v_2)$ resides in two polygons (Observation C-2). In this case, as illustrated in Fig. 7, removing $l(v_1, v_2)$ creates two polygons sharing a virtual edge (i.e., $l(v_1, v_2)$ becomes virtual), which according to Definition 3 need to be combined into a single polygon (line 10). (iii) $l(v_1, v_2)$ resides in one and only one polygon $\mathcal{D}$ (Observation C-3). This is the most complicated case, detailed below.

As Fig. 8 illustrates, removing $l(v_1, v_2)$ destroys $\mathcal{D}$ and splits the biconnected component $\mathcal{B}$ containing $\mathcal{D}$ into multiple biconnected components $\mathcal{B}_i$ $(i = 1, \ldots, \mu - 1)$, each corresponding to a subgraph of $\mathcal{B}$ separated from $\mathcal{D}$ by an edge $l(w_i, w_{i+1})$ in $\mathcal{D}$ ($w_1, \ldots, w_\mu$ are vertices in $\mathcal{D}$ defined in line 11). Accordingly, line 12 deletes $\mathcal{D}$ from the triconnected

Algorithm 5 Combine Polygon after Edge Deletion (CPED)

input : Triconnected decomposition Φ and a set of 2-vertex cuts F that are possibly between combinable polygons
output: Triconnected decomposition after combining polygons

```
1 foreach {w_i, w_{i+1}} in F do
2 |  if l(w_i, w_{i+1}) is a virtual edge and is only shared by
  |  two components D_{i1}, D_{i2} that are both polygons then
3 |  |  Φ ← combine D_{i1} and D_{i2} into a new polygon;
4 return updated Φ;
```

decomposition, and lines 13–21 update the decomposition of each $\mathcal{B}_i$ $(i = 1, \ldots, \mu - 1)$. If $\{w_i, w_{i+1}\}$ is not a 2-vertex cut, then $l(w_i, w_{i+1})$ must form a new bond, which creates a degenerate triconnected component (line 21). Otherwise, if $l(w_i, w_{i+1})$ is a virtual edge, then this edge may also disappear with the removal of $l(v_1, v_2)$. Specifically, as observed in Section VI-B, a virtual edge must be shared by at least two components. Therefore, if $l(w_i, w_{i+1})$ is only shared by $\mathcal{D}$ and another component $\mathcal{T}_i$ (which must be triconnected), then destroying $\mathcal{D}$ removes this edge. In this case (tested by line 15), line 16 recomputes the triconnected decomposition for the affected component (by the algorithm in [20]). Note that as in Case (2), the updated decomposition may create pairs of polygons separated by virtual edges of $\mathcal{T}_i$, in which case the auxiliary algorithm CPED is used to combine these polygons (lines 17). If $l(w_i, w_{i+1})$ is only shared by $\mathcal{D}$ and two other polygons, then destroying $\mathcal{D}$ causes these two polygons to be combined, again handled by CPED (line 19).

The correctness of Algorithm 4 is guaranteed by the following theorem.

Theorem 16: For any graph $\mathcal{G}$ and edge $l(v_1,\ v_2)$ in $\mathcal{G}$, given the triconnected decomposition of $\mathcal{G}$, TED returns the triconnected decomposition of $\mathcal{G} - l(v_1, v_2)$.

Complexity: The complexity of Algorithm 4 varies case by case: (i) If the deleted edge $l(v_1,\ v_2)$ is shared by at least three components or two components of which at least one is triconnected, then no update is performed. (ii) If $l(v_1,\ v_2)$ is in a bond or shared by exactly two polygons, then simple update is performed (lines 2–5 or line 10) in $O(1)$ time. (iii) If $l(v_1,\ v_2)$ lies within a triconnected component $\mathcal{T}$, then TED recomputes triconnected decomposition of $\mathcal{T}$ (lines 7–8), which takes $O(|V_{\mathcal{T}}|+|L_{\mathcal{T}}|)$ time [20]. (iv) If $l(v_1,\ v_2)$ is only contained by a polygon $\mathcal{D}$, then TED performs various updates on $\mathcal{D}$ and its neighboring components (lines 12–21), the most complicated update being for a triconnected component $\mathcal{T}_i$ separated from $\mathcal{D}$ by a virtual edge (lines 16–17), which takes $O(|V_{\mathcal{T}_i}|+|L_{\mathcal{T}_i}|)$ time similarly to case (iii). The overall complexity in this case is $O(\sum_{i=1}^{\mu-1}(|V_{\mathcal{T}_i}| + |L_{\mathcal{T}_i}|)) = O(|V_{\mathcal{B}}| + |L_{\mathcal{B}}|)$, where $\mathcal{B}$ is the biconnected component containing $l(v_1,\ v_2)$. Thus, the worst-case complexity of TED is $O(|V_{\mathcal{B}}| + |L_{\mathcal{B}}|)$.

Example: Fig. 8 illustrates one example of how Algorithm 4 updates the triconnected decomposition after deleting edge $l(v_1, v_2)$, which is handled by lines 12–21 of Algorithm 4. In this example, virtual edge $l(w_3, w_4)$ is removed after deleting $l(v_1, v_2)$, but virtual edge $l(w_1, w_2)$ is not (assuming all $\mathcal{T}_i$'s are triconnected components).

VII. Handling Practical Challenges

In practice, a monitoring system may face additional challenges. For example, what if we encounter an unpredicted topology at runtime? What if the computed placement requires too many monitors? What if the set of nodes can also change? We now discuss each of these issues and potential solutions.

A. On-Demand Monitor Placement

In case of unpredictable topology changes, existing monitors may not be able to identify all the links. One way to ensure identifiability is to employ additional nodes as *temporary* monitors, which only participate in taking measurements but not in other functions such as storage or processing of measurements. To distinguish from temporary monitors, we refer to monitors placed during network planning as *persistent* monitors. Given the current topology $\mathcal{G}_t$ and the persistent monitors M_0 (placed by the algorithm in Section IV), we can apply IMMP given in Algorithm 1 to select temporary monitors as needed to identify all the links in $\mathcal{G}_t$, and this is guaranteed to select the minimum number of temporary monitors by Corollary 4.

B. Bounded Number of Monitors

Given a bounded number of persistent monitors, we generally have to employ temporary monitors to achieve identifiability at runtime. For system stability, it is desirable to minimize the number of times we change node status (monitor→non-monitor or non-monitor→monitor). When the topology changes from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$, the minimum number of nodes with changed status is upper-bounded by the minimum number of extra monitors (in addition to persistent monitors) to identify both $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$, which is further upper-bounded by the minimum number of extra monitors to identify $\mathcal{G}_1, \ldots, \mathcal{G}_T$. The latter upper bound is minimized when the persistent monitors all belong to the optimal robust monitor placement. This observation suggests that we can adapt the algorithms in Section IV for placing a bounded number of persistent monitors by terminating the algorithms after selecting sufficient monitors.

C. Arrival/Departure of Nodes

We have limited topology changes to addition/removal of links, while in practice there may also be arrival/departure of nodes. We can handle arrivals by always selecting the newly arrived nodes as monitors; we can handle departures by treating each departure as removal of all the links incident to the departed node. It is easy to verify that this approach guarantees identifiability under node changes; we leave further optimization to future work.

VIII. Performance Evaluation

In this section, we first evaluate the efficiency of the proposed dynamic graph decomposition algorithm TED (Algorithm 4) in comparison with the best existing algorithm

TABLE II
DYNAMIC TOPOLOGIES FOR TAXI NETWORK (86 NODES)

range (m)	#topology changes	avg #links	avg #components
500	479	95.1	31.3
1000	479	334.3	6.3
1500	479	694.8	2.5
2000	479	1106.5	1.5
2500	479	1528.3	1.1
3000	479	1934.0	1.0
3500	479	2286.8	1.0

TABLE III
DYNAMIC TOPOLOGIES FOR TACTICAL NETWORK (90 NODES)

range (m)	#topology changes	avg #links	avg #components
15	399	325.9	17.3
75	293	539.9	10.4
225	196	1027.7	4.2
375	387	1256.1	2.0
450	380	1607.5	1.5
525	399	2191.1	1.1

that can handle edge deletions.[8] We then evaluate the proposed monitor placement algorithms (Section IV) based on dynamic topologies generated from mobility traces.

A. Performance of Dynamic Graph Decomposition

We evaluate TED against the best known algorithm [20] that can handle edge deletions on randomly generated dynamic topologies undergoing edge deletions. See Appendix B (in supplementary file) for details. The results show that TED outperforms the existing solution for a wide range of settings, and the improvement is the most significant for incremental changes in sparse graphs.

B. Performance of Monitor Placement Solutions

We now evaluate the proposed solutions on dynamic topologies generated from mobility traces. We use two datasets: (1) taxi cab traces from San Francisco,[9] from which we select traces of 86 nodes over a 8-hour period with location updates roughly every minute; (2) mobility traces generated by Rommie Hardy and Anjuli Smith at the Network Science Research Laboratory of the US Army Research Laboratory [31], which contain traces of 90 nodes belonging to 7 groups during a 400-second tactical operation with location updates every second.[10] Dataset (1) represents independent node mobility, and dataset (2) represents grouped node mobility.

We generate dynamic topologies from each trace by assuming a communication range and connecting two nodes by a link whenever they are within the range. See Tables II and III for a summary of the generated topologies. We see that both networks experience hundreds of topology changes.

Comparison of Algorithms: We compare the performance of the proposed robust monitor placement algorithms in terms of the number of monitors; see Fig. 9. We use greedy heuristics

[8]The case of edge insertions is handled by existing algorithms in [23], [24] and thus not evaluated here.

[9]Traces are available at: http://crawdad.org/epfl/mobility/.

[10]The anonymized traces used for this evaluation are available at: https://www.dropbox.com/s/bkhdifos9wzjwln/trace_90node_401second.txt?dl=0

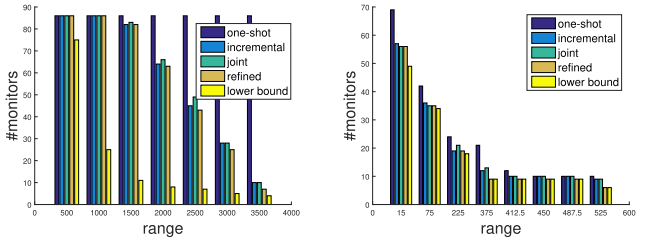


Fig. 9. Performance comparison under varying communication range. (a) taxi. (b) tactical.

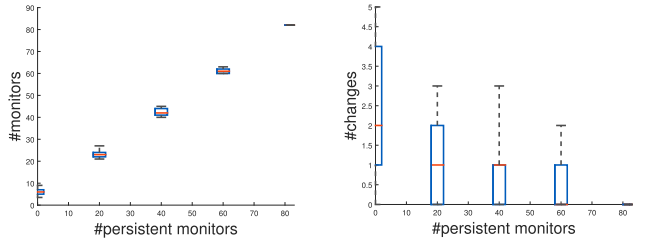


Fig. 10. Varying persistent monitors (taxi network, range = 1500 m). (a) #monitors. (b) #changes.

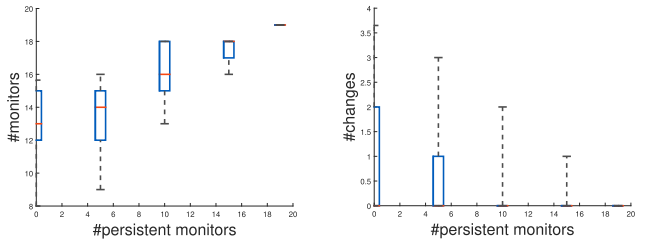
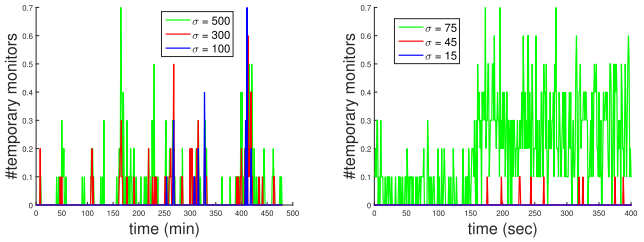


Fig. 11. Varying persistent monitors (tactical network, range=225 m). (a) # monitors. (b) # changes.

to solve the associated hitting set problems and the result of one-shot placement as the initial monitor set for refined placement. We also evaluate a lower bound on the number of monitors by computing the maximum number of monitors placed by MMP in any single topology. As expected, the one-shot placement algorithm uses the most monitors and the refined placement algorithm uses the fewest. Although incremental placement works well in these cases, we see that it can be improved by joint/refined placement. Note that the joint/refined placement here is suboptimal due to the greedy heuristics. Comparing the two networks, we see that as the taxi network has very different topologies during its lifetime, it requires a large number of monitors to maintain identifiability, while the tactical network contains subnets with relatively stable topologies and thus requires fewer monitors.

Varying Number of Persistent Monitors: Fig. 10-11 show the results when we bound the number of persistent monitors and place temporary monitors as needed to achieve identifiability, where the persistent monitors are randomly selected from monitors placed by the refined placement algorithm. To evaluate the cost of monitor adaptation, we count the number of changes in node status, i.e., a non-monitor becomes a temporary monitor or a temporary monitor becomes a non-monitor. Fig. 10 (a) shows the median (red bar),



(a) (b)

Fig. 12. Number of temporary monitors (averaged over 10 Monte Carlo runs) needed to achieve identifiability under prediction error. (a) taxi (range = 1500 m). (b) tactical (range = 225 m).

25/75-th percentile (box), and 5/95-th percentile (whisker) of the number of (both persistent and temporary) monitors over all topologies, and Fig. 10 (b) shows the corresponding values for the number of node status changes. The results show a clear tradeoff between the cost of monitors and the cost of adaptation, controlled by the number of persistent monitors.

Impact of Prediction Error: To evaluate how well our placement algorithms perform when input topologies contain errors, we add i.i.d. zero-mean Gaussian noise with variance σ^2 to node locations (x/y-coordinates) and treat the resulting topologies as the new ground truth; the process is repeated for multiple Monte Carlo runs. Meanwhile, the topologies computed from the original trace are provided to a monitor placement algorithm (refined placement) as predicted topologies.[11] We evaluate the robustness of the resulting placement by: (i) testing whether the placement achieves identifiability for each newly generated topology, and (ii) if not, computing the number of temporary monitors needed to achieve identifiability. As shown in Fig. 12, our monitor placement is highly robust to prediction error, requiring little help from temporary monitors even if the error in node location prediction[12] is as large as $1/3$ of the communication range. In fact, our placement achieves identifiability (i.e., not requiring any temporary monitor) for 0.95 fraction of time for the taxi network and 0.82 fraction of time for the tactical network (not shown).

IX. Conclusion

We have studied the problem of placing the minimum number of monitors to identify additive link metrics from end-to-end measurements in the presence of topology changes. Unlike existing solutions that consider a single topology, we want to simultaneously identify multiple topologies. By casting the problem as a generalized hitting set problem, we show that the optimal placement is NP-hard, but can be approximated to a logarithmic factor by the greedy heuristic and solved exactly in several special cases. We also develop a dynamic graph decomposition algorithm with sublinear complexity that assists the monitor placement algorithms but can also be used independently. Our evaluations on realistic dynamic topologies verify the efficiency and the robustness of the proposed solution.

[11]This is equivalent to treating the original topologies as ground truth and the modified topologies as estimates.

[12]The value of σ is basically the root-mean-squared error (RMSE) of the maximum likelihood estimate of node locations.

References

[1] T. He *et al.*, "Robust monitor placement for network tomography in dynamic networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[2] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *J. Amer. Statist. Assoc.*, vol. 91, pp. 365–377, 1996.

[3] A. B. Downey, "Using pathchar to estimate internet link characteristics," in *Proc. ACM SIGCOMM*, 1999, pp. 241–250.

[4] G. Jin, G. Yang, B. R. Crowley, and D. A. Agarwal, "Network characterization service (NCS)," in *Proc. IEEE HPDC*, Aug. 2001, pp. 289–299.

[5] E. Lawrence, G. Michailidis, V. N. Nair, and B. Xi, "Network tomography: A review and recent developments," in *Frontiers Statistics*. London, U.K.: Imperial College Press, 2006, pp. 345–364.

[6] O. Gurewitz and M. Sidi, "Estimating one-way delays from cyclic-path delay measurements," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1038–1044.

[7] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," in *Proc. ACM SIGCOMM*, 2004, pp. 55–66.

[8] A. Chen, J. Cao, and T. Bu, "Network tomography: Identifiability and Fourier domain estimation," in *Proc. IEEE INFOCOM*, May 2007, pp. 1875–1883.

[9] A. Gopalan and S. Ramasubramanian, "On identifying additive link metrics using linearly independent cycles and paths," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 906–916, Jun. 2012.

[10] L. Ma, T. He, K. K. Leung, A. Swami, and D. Towsley, "Inferring link metrics from end-to-end path measurements: Identifiability and monitor placement," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1351–1368, Aug. 2014.

[11] M. Zhao and W. Wang, "Analyzing topology dynamics in ad hoc networks using a smooth mobility model," in *Proc. IEEE WCNC*, Mar. 2007, pp. 3279–3284.

[12] I. W. Ho, K. K. Leung, and J. W. Polak, "Stochastic model and connectivity dynamics for VANETS in signalized road systems," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 195–208, Feb. 2011.

[13] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2307–2317.

[14] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," in *Proc. IEEE INFOCOM*, Oct. 2003, pp. 1092–1103.

[15] R. Kumar and J. Kaur, "Practical beacon placement for link monitoring using network tomography," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2196–2209, Dec. 2006.

[16] Y. Gao *et al.*, "Scalpel: Scalable preferential link tomography based on graph trimming," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1392–1403, Jun. 2016.

[17] L. Ma, T. He, K. K. Leung, A. Swami, and D. Towsley, "Monitor placement for maximal identifiability in network tomography," in *Proc. IEEE INFOCOM*, May 2014, pp. 1447–1455.

[18] W. Ren and W. Dong, "Robust network tomography:κ-identifiability and monitor assignment," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[19] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.

[20] J. E. Hopcroft and R. E. Tarjan, "Dividing a graph into triconnected components," *J. Comput.*, vol. 2, pp. 135–158, Sep. 1973.

[21] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification—A technique for speeding up dynamic graph algorithms," *J. ACM*, vol. 44, no. 5, pp. 669–696, Sep. 1997.

[22] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. ACM*, vol. 48, no. 4, pp. 723–760, 2001.

[23] G. D. Battista and R. Tamassia, "On-line maintenance of triconnected components with SPQR-trees," *Algorithmica*, vol. 15, no. 4, pp. 302–318, 1996.

[24] J. L. Poutré, "Maintenance of triconnected components of graphs," in *Proc. Int. Colloq. Automata, Lang., Programming (ICALP)*, 1992, pp. 354–365.

[25] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer, "Separator-based sparsification II: Edge and vertex connectivity," *J. Comput.*, vol. 28, no. 1, pp. 341–381, 1998.

[26] Y. Yao, W. Cai, V. Hilaire, A. Koukam, and C. Wang, "Statistical analysis technique on ad hoc network topology dynamic characteristics: Markov stochastic process," *Telecommun. Syst.*, vol. 53, no. 1, pp. 33–45, May 2013.
[27] S. Rajagopalan and V. Vazirani, "Primal-dual RNC approximation algorithms for (multi)-set (multi)-cover and covering integer programs," in *Proc. IEEE FOCS*, 1993, p. 322.
[28] G. Dobson, "Worst-case analysis of greedy heuristics for integer programming with non-negative data," *Math. Oper. Res.*, vol. 7, no. 4, pp. 515–531, Nov. 1982.
[29] N. Rug and A. Schobel, "Set covering with almost consecutive ones property," *Discrete Optim.*, vol. 1, pp. 215–228, Nov. 2004.
[30] M. Sakashita, K. Makino, and S. Fujishige, "Minimizing a monotone concave function with laminar covering constraints," *Discrete Appl. Math.*, vol. 156, no. 11, pp. 2004–2019, Jun. 2008.
[31] US Army Res. Lab. *The Network Science Research Laboratory*. [Online]. Available: https://www.arl.army.mil/www/default.cfm?page=2485

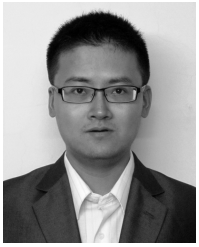
Ting He (S'04–M'07–SM'13) received the B.S. degree in computer science from Peking University, China, in 2003, and the Ph.D. degree in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2007.

From 2007 to 2016, she was a Research Staff Member with the Network Analytics Research Group, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. She is currently an Associate Professor with the School of Electrical Engineering and Computer Science, The Pennsylvania State University, University Park, PA, USA. Her work is in the broad areas of network modeling and optimization, statistical inference, and information theory.

Dr. He has received multiple paper awards from ITA, SIGMETRICS, ICDCS, IMC, and ICASSP, as well as the Research Division Award and the Outstanding Contributor Award from IBM.

Athanasios (Thanos) Gkelias (M'08–SM'15) received the M.Eng. degree in electrical and computer engineering (major in telecommunications) from the Aristotle University of Thessaloniki, Greece, in 2000, and the M.Sc. and Ph.D. degrees from the Department of Electronic Engineering, King's College London, in 2001 and 2006, respectively. He is currently a Post-Doctoral Researcher with the Imperial College London, where he is involved in the International Technology Alliance Project. He is also serving as the Project Manager of the UDRC Phase-I Follow-on work framework.

From 2009 to 2013, he served as the Project Manager with the University Defence Research Centre in Signal Processing, Imperial College, sponsored by the U.K. Ministry of Defence. In 2008, he was with the Bell-Labs Research Centre, Alcatel-Lucent, U.K., where he was a Visiting Researcher on wireless mesh networks. In 2007, he was a Visiting Researcher with Athens Information Technology, Greece.



Liang Ma received the B.Sc. and M.Sc. degrees from the Beijing University of Posts and Telecommunications, China, in 2007 and 2010, respectively, and the Ph.D. degree from the Imperial College London, U.K., in 2014. He was with NTT DoCoMo Beijing Laboratories, Ericsson Communications, China, and Microsoft Research Asia, where he was involved in WLAN medium access control, high-speed switching system, and software radio-based gigabit multi-antenna communications, respectively. He is currently a Research Staff Member with the Department of Cognitive Distributed Systems, IBM T. J. Watson Research Center, NY, USA. He was a recipient of the International Conference on Distributed Computing System Best Paper Award, the IBM Patent Award 2013, the Best Student Paper Award of ITA in Network & Information Sciences 2013, the ACM Internet Measurement Conference Best Paper Award Nomination, the INFOCOM 2014 Student Travel Grant, and the winner of Outstanding Graduate Student 2008 and Excellent Student Awards four times from 2003 to 2006.

Kin K. Leung (S'78–M'86–SM'93–F'01) received the B.S. degree from the Chinese University of Hong Kong in 1980, and the M.S. and Ph.D. degrees from the University of California at Los Angeles, Los Angeles, CA, USA, in 1982 and 1985, respectively. He joined AT&T Bell Labs, NJ, USA, in 1986 and worked at its successors, AT&T Labs and Lucent Technologies Bell Labs, until 2004. Since 2004, he has been the Tanaka Chair Professor with the Electrical and Electronic Engineering (EEE) and the Computing Departments, Imperial College, in London, U.K. He is currently the Head of the Communications and Signal Processing Group, EEE Department. His current research focuses on protocols, optimization, and modeling of various wireless networks. He also works on multiantenna and cross-layer designs for these networks. He received the Distinguished Member of Technical Staff Award from AT&T Bell Labs (1994), and was a co-recipient of the Lanchester Prize Honorable Mention Award (1997). He received the Royal Society Wolfson Research Merits Award (2004–2009) and became a member of Academia Europaea (2012). He also received several best paper awards, and actively served on conference committees and as journal editors.

Ananthram Swami (M'79–SM'96–F'08) received the B.Tech. degree from IIT Bombay, the M.S. degree from Rice University, and the Ph.D. degree from the University of Southern California (USC), all in electrical engineering. He is with the U.S. Army Research Laboratory and is the Army's Senior Research Scientist (ST) for Network Science. Prior to joining ARL, he held positions with Unocal Corporation, USC, CS-3, and Malgudi Systems. He was a Statistical Consultant to the California Lottery and developed a MATLABatlab-based toolbox for non-Gaussian signal processing. He has held visiting faculty positions at INP, Toulouse, France, and currently at Imperial College. His work is in the broad area of network science, with emphasis on tactical communication networks. He is an ARL Fellow.

Don Towsley (M'78–SM'93–F'95–LF'15) received the B.A. degree in physics and the Ph.D. degree in computer science from the University of Texas in 1971 and 1975, respectively. He held visiting positions with numerous universities and research labs. He is currently a Distinguished Professor with the Department of Computer Science, University of Massachusetts. His research interests include networks and performance evaluation.

He was a founding Co-Editor-in-Chief of the new *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* and has served as the Editor-in-Chief of the IEEE/ACM TRANSACTIONS ON NETWORKING and on numerous editorial boards. He has served as the Program Co-Chair of several conferences, including INFOCOM 2009.

Dr. Towsley has been an elected Fellow of both the ACM and IEEE and is a corresponding member of the Brazilian Academy of Sciences. He has received numerous IEEE and ACM awards, including the 2007 IEEE Koji Kobayashi Award and the ACM SIGCOMM and ACM SIGMETRICS Achievement Awards.