This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

Distributed Calculation of Edge-Disjoint Spanning Trees for Robustifying Distributed Algorithms against Man-in-the-Middle Attacks

Gabriele Oliva*, Sebastian Cioabă† and Christoforos N. Hadjicostis‡

***Abstract*—In this paper we provide a distributed methodology to allow a network of agents, tasked to execute a distributed algorithm, to overcome Man-in-the-middle attacks that aim at steering the result of the algorithm towards inconsistent values or dangerous configurations. We want the agents to be able to restore the correct result of the algorithm in spite of the attacks. To this end, we provide a distributed algorithm to let the set of agents, interconnected by an undirected network topology, construct several *edge-disjoint spanning trees* by assigning labels to their incident edges. The ultimate objective is to use these spanning trees to run multiple instances of the same distributed algorithm in parallel, in order to be able to detect Man-in-the-middle attacks or other faulty or malicious link behavior (e.g., when the instances yield different results) and to restore the correct result (when the majority of instances is unaffected). The proposed algorithm is lightweight and asynchronous, and is based on iterated depth-first visits on the graph. We complement the paper with a thorough analysis of the performance of the proposed algorithms.**

Keywords: Distributed Algorithms, Man-in-the-middle Attacks, Tree Packing Problem, Edge Disjoint Spanning Trees.

I. Introduction

Distributed algorithms, such as consensus algorithms [1], [2], have been proven quite effective in allowing a set of interconnected agents to perform complex global computations, optimizations, or other desirable operations, by means of iterative, simple and local interactions among neighboring agents. These algorithms, however, are often prone to failures and malicious data manipulations, and there is a need to provide reliable methodologies to overcome such issues.

Several solutions have been provided in the literature. For example, in [3] a distributed Byzantine consensus algorithm that tolerates message omissions by a subset of the agents is provided; in [4] a related methodology for directed graphs is developed; in [5] a framework that places emphasis on the trust among the nodes is given. More recently, an adaptive finite-time approach has been provided in [6], while a methodology that can overcome actuator faults is presented in [7]; moreover, in [8] an approach that can handle a variable network structure is proposed.

The above approaches focus on the agent's behavior, while malicious attacks affecting the links are typically neglected; there is, however, a relevant literature on distributed algorithms with packet drops, especially in the case of average consensus algorithms: for instance in [9] an approach to obtain a value close to the nominal one is provided, under certain conditions, while the algorithms provided in [10], [11] achieve exact consensus in spite of packet losses.

* Complex Systems & Security Laboratory, University Campus Bio-Medico, via A. del Portillo 21, 00128, Rome, Italy.
Email g.oliva@unicampus.it. Corresponding author.
† Department of Mathematical Sciences, University of Delaware, Newark, DE 19716-2553. Email cioaba@udel.edu. This research was partially supported by NSF grant DMS-1600768.
‡ Department of Electrical and Computer Engineering, University of Cyprus, 75 Kallipoleos Avenue, P.O. Box 20537, 1678 Nicosia, Cyprus.
Email chadjic@ucy.ac.cy.

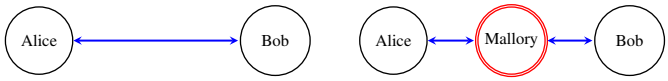

Fig. 1. Example of Man-in-the-middle attack: Mallory intercepts all communications between Alice and Bob. As a result of the attack, Mallory can modify the content of messages exchanged between Alice and Bob, without being noticed.

Among other issues, *Man-in-the-middle* (MITM) attacks [12]–[14] represent an effective way to alter the data transmitted by the agents, potentially steering the network towards inconsistent results or dangerous configurations. As shown in Figure 1, MITM attacks consist of an attacker (Mallory) becoming the proxy for the communication between two victim nodes (Alice and Bob). Specifically, Mallory intercepts all messages between Alice and Bob and it sends maliciously altered messages to Alice pretending to be Bob, and vice-versa. In this way, Alice and Bob believe they are directly communicating with each other, and are unable to detect the attack.

A. Related Work: Fault-Tolerant Distributed Algorithms

In the literature there have been attempts to develop distributed algorithms and protocols having some resistance to MITM attacks. In [15] Chiang et al. develop a methodology for distributed time synchronization that has some robustness to MITM attacks, in that only limited damage can be dealt by malicious attackers, which can cause only constant delays. In [16] a secure time synchronization protocol was developed, based on pairing and identity-based cryptography. In [17] received signal strength information is used to spot spoofed messages. Such approaches, however, require the implementation of sophisticated countermeasures, which are often tailored to the specific application (e.g., time synchronization).

Note that fault-tolerant message delivery protocols are often implemented by constructing multiple paths on the network [18]–[20], by requiring the existence of such disjoint paths [21], or by building independent subnetworks [22], [23]. In this paper we follow such a perspective while facing MITM attacks; our focus, therefore, is not on preventing the occurrence of MITM attacks, but on guaranteeing that the distributed algorithm executed by the agents achieves its purpose, in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

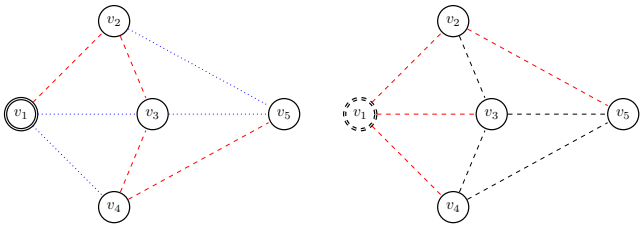

Fig. 2. Comparison between depth-first visit (left plot) and breadth-first visit (right plot) starting from node v_1. In the first case we get 2 EDSTs (red dashed and blue dotted edges), while in the second case we obtain a single spanning tree. However, the diameter obtained for the breadth-first visit is considerably smaller than the diameter of the EDSTs obtained via the depth-first visit.

spite of successful MITM attacks. To achieve this result, we partition the network into several *Edge-Disjoint Spanning Subgraphs* (EDSSGs), i.e., connected subgraphs spanning all the nodes and not having common edges. Once these EDSSGs are defined, the agents run independent instances of the same distributed algorithm over each EDSSG. By comparing the results obtained by the different instances of the algorithm, which are supposed to yield the same result under nominal conditions, the agents are able to detect the attack (e.g., when the results obtained are not all the same) and to restore the correct value (when the majority of instances are unaffected).

To achieve this result in a distributed way, we develop a methodology to let the agents in the network partition their incident links in order to construct several *Edge-Disjoint Spanning Trees* (EDSTs) over the network, with the aim to use these EDSTs as the backbone for the EDSSGs.

B. Related Work: Edge-Disjoint Spanning Trees

The problem of finding the maximum number of EDSTs over a graph, often referred to as the *Tree Packing Problem* has been faced since the beginning of the 1960's when mathematicians like Tutte [24] and Nash-Williams [25] provided the same combinatorial conditions to calculate the maximum number of EDSTs. Other theoretical results include a shorter proof of the Tutte-Nash-Williams condition [26], and a characterization of the number of spanning trees based on the eigenvalues of the adjacency or Laplacian matrices of regular graphs [27], further extended to general graphs in [28]. In [29] the authors characterize the expected minimum total weight of the number of EDSTs over complete weighted graphs.

From an algorithmic point of view, there are several approaches for obtaining EDSTs in the literature, and they are intrinsically centralized. The centralized algorithm in [18] calculates two EDSTs; the algorithm presented in [30] is based on the work of Edmonds on *matroid theory* [31] and is able to construct the maximum number of EDSTs in a graph in polynomial time. Available distributed approaches are limited to extremely particular structures, such as hypercubes [32] or twisted cubes [33].

C. Contribution and Paper Outline

In this paper we provide a lightweight and asynchronous, yet suboptimal, algorithm that iteratively lets the agents label their incident links as part of different EDSTs. The key idea behind the algorithm is that, in order to increase the chances to have several edge-disjoint spanning trees, a good strategy is to resort to iterated depth-first visits of the graph, each time removing edges that have been labeled (i.e.,used) in previous iterations.

The procedure is iterated as long as it is possible, i.e., until the remaining links form a disconnected graph, as shown in the left plot of Figure 2, where 2 EDSTs are found. Other approaches, such as a breadth-first visit (see the right plot in Figure 2) are not suitable for the problem at hand, as in the breadth-first visit all links that are incident to the starting node are used for constructing the first EDST, thus preventing the construction of additional EDSTs.

The proposed approach, apart from constructing several EDSTs, also allows the nodes to calculate useful meta-information such as the diameter of each EDST and the number of nodes in the network.

The outline of the paper is as follows: Section II provides some preliminary definitions; Section III introduces the problem at hand and discusses the motivations underlying the proposed approach; Section IV collects some classical theoretical and algorithmic results related to finding the maximum number of EDSTs in a graph; Sections V and VI introduce the proposed algorithms for finding a single spanning tree, and several EDSTs, respectively; Section VII provides a simulation campaign aimed at assessing the performances of the proposed approach, while some conclusions and future work directions are collected in Section VIII.

II. Preliminaries

Let $G = \{V, E\}$ be a *graph* with n nodes $V = \{v_1, v_2, \dots, v_n\}$ and e edges $E \subseteq V \times V$, where $(v_i, v_j) \in E$ captures the existence of a communication link from node v_i to node v_j. A graph is said to be *undirected* if $(v_i, v_j) \in E$ whenever $(v_j, v_i) \in E$, and is said to be *directed* otherwise.

A *path* P over a graph $G = \{V, E\}$ is a sequence of nodes $v_{i_1}, \dots, v_{i_t}$ such that $(v_{i_k}, v_{i_{k+1}}) \in E$ for $k = 1, 2, ..., t-1$; the *length* of the path is the number of edges involved $(t-1)$ and it is denoted by $|P| = t-1$. A *cycle* is a path of nonzero length that starts and ends at the same node, while a path is called *simple* if it does not contain cycles. A *minimum path* between v_i and v_j is a simple path from v_i to v_j with minimum length. An undirected graph is *connected* if for each pair of nodes v_i, v_j there is a simple path over G that connects them. An *acyclic graph* is a graph without cycles.

In the following, although some of the results discussed in Section IV apply to *multigraphs* (i.e., graphs with multiple instances of an edge between the same pair of nodes) and to graphs with *self-loops* (i.e., graphs containing links in the form (v_i, v_i)), we will consider undirected connected graphs without self-loops or multiple edges.

Let the *neighborhood* $\mathcal{N}_i$ of node v_i be the set of nodes v_j such that $(v_j, v_i) \in E$. The *degree* d_i of a node v_i is the number of its edges, i.e., $d_i = |\mathcal{N}_i|$. Let P_{ij} be the minimum

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

path connecting node v_i to node v_j, with $v_i, v_j \in V$. The eccentricity of node $v_i \in V$ is given by

$$\epsilon_i = \max_{v_j \in V \setminus \{v_i\}} \{|P_{ij}|\},$$

i.e., it is the maximum length among the minimum paths connecting v_i to each of the remaining nodes. The *diameter* δ of a graph G is given by

$$\delta = \max_{v_i \in V} \left\{ \max_{v_j \in V \setminus \{v_i\}} \{|P_{ij}|\} \right\} = \max_{v_i \in V} \{\epsilon_i\},$$

i.e., it is the length of the longest minimum path between any pair of nodes in V.

A graph can be represented via an $n \times n$ *adjacency matrix* A, whose entries $a_{ij} = 1$ if $(v_j, v_i) \in E$ and $a_{ij} = 0$ otherwise. The *degree matrix* D is a diagonal matrix whose diagonal elements are

$$d_{ii} = \sum_{j=1}^{n} a_{ij} = |\mathcal{N}_i|.$$

The *Laplacian matrix* is $L = D - A$, while the *signless Laplacian matrix* is $L_s = D + A$. Given a graph G, we denote by $\lambda_i(G)$, $\mu_i(G)$ and $\theta_i(G)$ the i-th smallest eigenvalue of A, L and L_s, respectively[1].

A *tree* T is a connected acyclic graph. We say T is *rooted* at a node v_i if v_i has been designated the *root* of T. With respect to a root node v_i, a *leaf* is a node $v_j \neq v_i$ such that $d_j = 1$, while the *depth* of a node v_j is the length of the path over T from the root node v_i to v_j. Given a node v_j in a tree and the root v_i, the *branch* of the tree for node v_j is the set of nodes that are in a path P, from v_j to a leaf node, such that $v_i \notin P$. A *spanning tree* of a graph $G = \{V, E\}$ is a connected acyclic subgraph $T = \{V, E_T\}$, where $E_T \subseteq E$. A graph composed of one of more connected components, each being a tree, is referred to as a *forest*.

Two subgraphs $G_1 = \{V, E_1\}$ and $G_2 = \{V, E_2\}$ of a graph $G = \{V, E\}$ are *edge-disjoint* if $E_1 \cap E_2 = \emptyset$. In particular, if G_1 and G_2 are connected they are *edge-disjoint spanning subgraphs* (EDSSG). Similarly, we refer to two connected and edge-disjoint trees $T_1 = \{V, E_{T_1}\}$ and $T_2 = \{V, E_{T_2}\}$ as *edge-disjoint spanning trees* (EDST).

III. Problem Statement and Motivation

Consider a set of n agents interconnected by an undirected and connected graph $G = \{V, E\}$ and suppose the agents need to exchange information in order to implement a distributed algorithm $\mathcal{A}$ that satisfies the following assumption.

***Assumption** 1:* The distributed algorithm $\mathcal{A}$ achieves a steady-state result (asymptotically or in finite time) for each agent (not necessarily the same result for all agents) which is invariant with respect to the underlying graph G, as long as it is connected and undirected.

Note that algorithm $\mathcal{A}$ might be prone to data manipulations such as man-in-the-middle attacks, where malicious attackers hijack the communication between one or more pairs of nodes, modifying the information sent, without being noticed. As a consequence of the attack, algorithm $\mathcal{A}$ might yield completely different results at each agent, preventing them from performing their intended functions.

Our objective is therefore to endow the agents with the ability to detect MITM attacks and recover the correct result of the algorithm in spite of the attack. To this end, we note that if the graph G has been partitioned into several EDSSGs and the agents execute, in parallel, an instance of algorithm $\mathcal{A}$ over each EDSSG, then malicious manipulations can be spotted after the termination of the instances of $\mathcal{A}$ (or after the transients are extinguished when $\mathcal{A}$ exhibits asymptotic convergence).

Specifically, by running in parallel the desired distributed algorithm $\mathcal{A}$ over m EDSSGs, each agent computes a set of m steady-state values or results. Note that, if the algorithm being executed has asymptotic convergence, the agents may compute an approximation of the asymptotic values, using stopping criteria like those in [34], [35] and references therein, thus obtaining values that are within a small distance from the asymptotic values. One can also use finite time approaches like the ones in [21], [36]. Under nominal conditions, all such values must be the same (or must have negligible discrepancies in the case of asymptotic algorithms). In the event of man-in-the-middle attacks affecting the links of a subset of the EDSSGs, the agents are able to detect the problem if there are discrepancies in the result of the different instances. Moreover, if the majority of EDSSGs is not affected by the attack, the agents are able to determine the correct result. However, this strategy is effective only after the execution of the algorithm (or after transients are extinguished in the case of asymptotic algorithms).

Moreover, we point out that in the proposed framework the agents are able to detect that one or more steady-state values are corrupted or differ from the majority of the steady state values, but are not able to determine the exact links that are under MITM attack. Instead, by detecting that one final value is corrupted or differs from the majority of the other values, the agents are able to conclude that the links under attack must belong to the spanning subgraph associated to that particular final value, and may decide not to use the spanning subgraph for further computations.

We now elaborate on the main motivations for our problem setting in the following remarks, then we introduce the problems studied in this paper and we discuss a strategy to address them.

***Remark** 1:* As long as at least one final value is different from the others, within the proposed strategy the agents are able to detect a man-in-the-middle attack, even if they might not be able to retrieve the correct result for Algorithm $\mathcal{A}$. In other words, the agents are guaranteed to be able to spot an attack if at most $m - 1$ links in E are attacked.

[1]Since these matrices are symmetric, their eigenvalues are real-valued.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

***Remark** 2:* Within the proposed strategy, Algorithm $\mathcal{A}$ is robust with respect to man-in-the-middle attacks leaving the majority of the EDSSGs unaffected. In other words, the agents are guaranteed to be able to compute the correct result if at most ζ links in E are attacked, with

$$\zeta = \left\lceil \frac{m}{2} \right\rceil - 1.$$

***Remark** 3:* Note that, when at least $\zeta+1$ links are affected, there is the possibility that the majority of the results is faulted, preventing the agents from calculating the right values. However, attacking more than ζ links is not sufficient to have a guarantee to affect the final result. This happens because the attacker needs to attack at least $\zeta+1$ links, each belonging to a different EDSSGs. To succeed in this task, the attacker needs either prior knowledge on the network or enough resources to deal a large-scale attack.

***Remark** 4:* In order to drive the agents towards incorrect final values, the altered values must all be the same. This, again, implies that, depending on the particular algorithm being executed, the attacker may need to spend nontrivial effort in order to successfully steer the algorithm towards a different result.

For the reasons collected in Remarks 1–4, in this paper we are interested in solving the following problem.

***Problem** 1:* Given a connected and undirected graph $G = \{V, E\}$ find a maximum number of EDSSGs.

In this paper we are interested in solving the problem in a distributed way, i.e., we want each agent to interact with its neighbors in a distributed way in order to label their incident edges with different labels, so that links with the same label belong to the same EDSSG.

Note that finding a maximum number of EDSSGs essentially coincides with finding a maximum number of EDSTs. Moreover, solving exactly Problem 1, especially in a distributed way, might be impractical because, as discussed in the next section, available solutions in the literature are essentially centralized. In the remainder of this paper, therefore, we adopt a strategy which is suboptimal but implementable in a distributed context. Specifically, we provide an approximate solution to Problem 1 by iteratively performing depth-first visits of the graph and by removing the visited links from consideration, thus building multiple EDSTs.

IV. Maximum Number of EDSTs

In this section we discuss some results on the maximum number $\tau(G)$ of edge-disjoint spanning trees of a graph G.

A. Combinatorial Condition

The following theorem, proved independently by Tutte and Nash-Williams [24], [25], provides a combinatorial condition for finding $\tau(G)$.

***Theorem** 1:* An undirected multi-graph $G = (V, E)$ contains k edge-disjoint spanning trees *iff* for every partition Π of V into l sets, $V_1, V_2, \ldots, V_l$, the number q of edges having endpoints in different sets of the partition Π is at least $q = k(l-1)$.

***Remark** 5:* The direct application of the above condition to calculate $\tau(G)$ is rather impractical, because of the need to inspect a vast number of partitions. In fact, for n nodes, the total number of partitions of the nodes is given by the *Bell number* B_n, which can be obtained by taking $B_0 = B_1 = 1$ and by applying the recursive rule, for $m \geq 2$

$$B_{m+1} = \sum_{k=0}^{m} \binom{m}{k} B_k.$$

In [37] it is shown that

$$B_m \approx \frac{1}{\sqrt{m}} \lambda(m)^{m+1/2} e^{\lambda(m)-m-1},$$

where $\lambda(m)$ is such that $\lambda(m)\log(\lambda(m)) = m$.

B. Optimal but Centralized Algorithm

In [30], Roskind and Tarjan provide an efficient, yet centralized, algorithm to find a maximum number of EDSTs in a graph, based on *matroid theory*.

Let a set X of n elements and define some of the subsets of X as *independent*. The set X is a *matroid* if the independence of any $S \subseteq X$ implies the independence of any subset of S. Matroids have the interesting property that the largest independent subset can be found by means of greedy algorithms [31]. In [30] the authors cast the problem at hand in terms of matroids: given a graph $G = \{V, E\}$, they choose $X = E$ and they say $E' \subseteq E$ is independent with respect to an integer k if $G' = \{V, E'\}$ can be partitioned into k edge-disjoint forests. The main idea of the algorithm in [30] is, therefore, to iteratively grow a set of k edge-disjoint forests $F = \{F_1, \ldots, F_k\}$, considering a link $e \in E$ at each time and checking the independence of $F \cup \{e\}$. If $F \cup \{e\}$ is independent, then the algorithm modifies the assignment of some of the links in F and adds e to F. If, conversely, $F \cup \{e\}$ is dependent, then the algorithm increases the number k of forests (initially, $k = 1$).

The above algorithm, therefore, invokes $|E|$ times an *oracle*, that is, a function that is able to tell the independence of $F \cup \{e\}$ (and to reorganize the links in the forests to avoid cycles). The oracle is the bottleneck of the approach, and in [30] an efficient implementation is provided, which has a computational complexity $O(|E|^2)$; the implementation is based on the fast disjoint set union algorithm [38]. Such an approach, however, is centralized and appears hard to translate in a distributed context.

C. Lower bound on $\tau(G)$

The following Theorem provides a lower bound on $\tau(G)$ [28], which extends the result given in [27]. We will take advantage of this lower bound in Section VII while discussing the performances of the proposed distributed algorithms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

***Theorem** 2 (Liu et al., 2014 [28]):* Let G be a graph of minimum degree $d_{\min} \geq 2\tau_{lb}(G) \geq 4$; the following conditions hold true:

(1) If $\lambda_{n-1}(G) < d_{\min} - \frac{2\tau_{lb}(G)-1}{d_{\min}+1}$, then $\tau(G) \geq \tau_{lb}(G)$.
(2) If $\theta_{n-1}(G) < 2d_{\min} - \frac{2\tau_{lb}(G)-1}{d_{\min}+1}$, then $\tau(G) \geq \tau_{lb}(G)$.
(3) If $\mu_2(G) > \frac{2\tau_{lb}(G)-1}{d_{\min}+1}$, then $\tau(G) \geq \tau_{lb}(G)$.

In [27] it is shown that the above bounds are tight in the case of regular graphs with $\tau_{lb}(G) = 2$ or $\tau_{lb}(G) = 3$, while the result is extended to graphs with $\tau_{lb}(G) \geq 4$ in [39].

V. Distributed Construction of EDSTs

As discussed above, the algorithm from Roskind and Tarjan is quite hard to implement in a distributed fashion. In this paper we present a distributed methodology for obtaining EDSTs based on repeated depth-first visits of the graph; in this section we develop an algorithm to construct a single spanning tree, while in the next section we discuss the construction of multiple EDSTs.

Note that the algorithm we present in the remainder of this section exploits the typical token-passing approach that implements a depth-first visit in a graph, which is quite diffused in the literature of distributed algorithms [40]–[43]. With respect to the state of the art, the main improvement here is the ability to calculate the diameter of the tree while it is being built. As discussed in the conclusive section, this feature can be the basis to guide the construction of EDSSGs with small diameter; specifically, it might be useful to sacrifice some of the EDSTs in order to have more "spare links" to be assigned to the remaining EDSTs, with the aim to reduce their diameter and boost the convergence time of the distributed algorithms being executed.

A. Distributed Spanning Tree Construction Algorithm

In this section we develop a distributed algorithm to let a set of agents, interconnected by an undirected and connected graph, find a spanning tree by labeling some of the edges; we call this algorithm the *Distributed Spanning Tree Construction Algorithm* (DSTC). The pseudocode of the proposed DSTC Algorithm is reported in Algorithm 1, while an example of its execution is shown in Figure 3. The algorithm amounts to a depth-first visit of the graph, starting from a leader node v_{i*}, which can be elected via several techniques (e.g., see [44] and references therein). Specifically, we assume the depth-first visit is implemented in an asynchronous way, by letting the nodes pass each other a single *token* (initially the leader has the token). The node holding the token passes it to a neighbor node which has not yet been visited (if any), otherwise it returns the token to its *father* (i.e., to the node from which it received the token in first place). The spanning tree is constructed by selecting the edges through which the token is passed among the nodes. In the process, each node receiving a token also updates some internal variables, which are used to keep track of the links in the spanning tree and to calculate useful meta-information such as the number of nodes and the diameter of the spanning tree that is being constructed. The procedure

Algorithm 1 Distributed Spanning Tree Construction Algorithm (DSTC)

procedure INITIALIZATION(ID,is-leader)
  $v_i^{father} \leftarrow \emptyset$;
  $\mathcal{M}_i \leftarrow \mathcal{N}_i$;
  $\Delta^\dagger \leftarrow 0$; // biggest Δ from v_i to a leaf
  $\Delta^\ddagger \leftarrow 0$; // second biggest Δ from v_i to a leaf
  $\delta_T^\dagger \leftarrow 0$; // estimate of the diameter
  label (v_i, v_j) with $-\infty$ for all $v_j \in \mathcal{N}_i$;
  if is-leader **then**
    visited$_i \leftarrow 1$;
    send $< i$,visited$_i >$ to all neighbors;
    $\Delta_i \leftarrow 0$;
    select random $v_j \in \mathcal{M}_i$;
    send token $<$ ID$, 1, 0, 0, 0 >$ to v_j;
    label (v_i, v_j) with ID;
  else
    visited$_i \leftarrow 0$;
    $\Delta_i \leftarrow \infty$; // depth of v_i in the tree
procedure ONRECEIVEVISITED($< j$,visited$_j >$)
  $\mathcal{M}_i \leftarrow \mathcal{M}_i \backslash \{v_j\}$;
procedure ONRECEIVETOKEN(v_s,$<$ ID$, m, \Delta, \Delta_s, \delta_T >$)
  label (v_s, v_i) with ID;
  if $\Delta_i = \infty$ **then** // update Δ_i on first token received
    $\Delta_i \leftarrow \Delta_s + 1$;
    $v_i^{father} \leftarrow v_s$;
  if $\Delta > \Delta^\dagger$ **then** // update $\Delta^\dagger$ and $\Delta^\ddagger$
    $\Delta^\ddagger \leftarrow \Delta^\dagger$;
    $\Delta^\dagger \leftarrow \Delta$;
  else if $\Delta > \Delta^\ddagger$ **then**
    $\Delta^\ddagger \leftarrow \Delta$;
  if $\delta_T > \delta_T^\dagger$ **then** // update $\delta_T^\dagger$
    $\delta_T^\dagger \leftarrow \delta_T$;
  $\overline{\delta}_T \leftarrow \max\{\delta_T^\dagger, \Delta^\dagger + \Delta^\ddagger - 2\Delta_i\}$;
  /* Send token */
  if $\mathcal{M}_i \neq \emptyset$ **then** //send towards leaves
    select random $v_j \in \mathcal{M}_i$;
    if is-leader **then**
      send $< ID, m, 0, 0, 0 >$ to v_j;
    else if not visited$_i$ **then**
      visited$_i \leftarrow 1$;
      send $< i$,visited$_i >$ to all neighbors;
      send $< ID, m+1, \Delta+1, \Delta_i, 0 >$ to v_j;
    else
      send $< ID, m, \Delta_i, \Delta_i, 0 >$ to v_j;
  else if not $v_i^{father} = \emptyset$ **then** // send towards root
    if not visited$_i$ **then**
      visited$_i \leftarrow 1$;
      send $< i$,visited$_i >$ to all neighbors;
      send $< ID, m+1, \Delta+1, \Delta_i, \overline{\delta_T} >$ to v_i^{father};
    else
      send $< ID, m, \Delta^\dagger, \Delta_i, \overline{\delta_T} >$ to v_i^{father};
    stop;
  else // v_i is the leader and visit complete
    $\delta \leftarrow \overline{\delta}_T$;
    stop;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

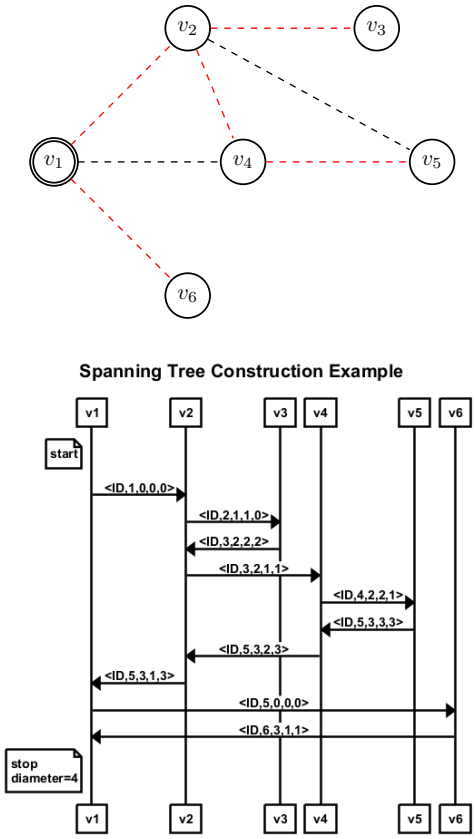

Fig. 3. Example of execution of the proposed DSTC algorithm. Red dotted links are labeled as belonging to a spanning tree.

is completed when the token returns to the leader and all its neighbors have been visited.

The token contains some information about the spanning tree that is being constructed, and some of the contents of the token are modified at each passage. Specifically, the token is the 5-tuple

$$< \text{ID}, m, \Delta, \Delta_s, \delta_T >$$

where

- ID is the identifier of the spanning tree being constructed;
- m is the current value of the number of nodes visited so far;
- Δ is the depth of the spanning tree so far (i.e., the maximum length of a path connecting a visited node to the leader);
- Δ_s is the depth of the node v_s that sends the token;
- δ_T is the current estimate of the diameter of the spanning tree.

In addition to passing the token, each node v_i maintains and updates the following parameters during the execution of the algorithm:

- a binary label visited$_i$ that is used to determine if v_i has been visited (we initialize visited$_i = 0$ for all nodes);
- the identifier of its father over the tree (initialized to zero);
- an integer Δ_i that represents the depth of v_i over the spanning tree–initially $\Delta_i = \infty$ for all nodes but the leader, which sets $\Delta_i = 0$;
- the set $\mathcal{M}_i$ of not visited neighbors, which is initially equal to its neighborhood $\mathcal{N}_i$;
- the labels associated to the links (v_i, v_j) towards its neighbors v_j (initialized to $-\infty$);
- two integers $\Delta^\dagger$ and $\Delta^\ddagger$, which are the biggest and the second biggest depths of the nodes that branch from v_i along the tree (both initialized to zero for all nodes);
- an integer $\delta^\dagger$ which is an estimate of the diameter of the tree (initialized to zero for all nodes).

During the initialization phase, the leader v_{i*} sends the token

$$< ID, 1, 0, 0, 0 >$$

to a random node $v_j \in \mathcal{M}_{i*}$, and it labels (v_{i*}, v_j) with the identifier ID of the current spanning-tree; then the leader becomes visited and it provides such an information to all its neighbors, by broadcasting the message

$$< i^*, \text{visited}_{i^*} > .$$

Every time a node v_i receives information about a neighbor v_j that becomes visited, v_j is removed from $\mathcal{M}_i$.

When a node v_i receives a token for the first time, it updates its depth setting $\Delta_i = \Delta + 1$, while each time a node receives a token, it updates the value of the biggest depth $\Delta^\dagger$ and second biggest depth $\Delta^\ddagger$ of one of its neighbors that branch from v_i along the tree; then, node v_i updates $\delta^\dagger$ and it calculates

$$\overline{\delta}_T = \max\{\delta^\dagger_T, \Delta^\dagger + \Delta^\ddagger - 2\Delta_i\}.$$

The node v_i attempts to transmit the token downwards the tree to a not visited neighbor $v_j \in \mathcal{M}_i$. If such a neighbor exists, we have three cases:

1) if v_i is the leader, then it sends to node v_j the token
$$< ID, m, 0, 0, 0 >;$$
2) if v_i is not visited, then it becomes visited, it informs its neighbors and it sends to node v_j the token
$$< ID, m + 1, \Delta + 1, \Delta_i, 0 >;$$
3) otherwise v_i sends to node v_j the token
$$< ID, m, \Delta_i, \Delta_i, 0 > .$$

If $\mathcal{M}_i = \emptyset$, conversely, node v_i transmits the token upwards to its parent, unless v_i is the leader. Specifically, three cases are possible:

I) if v_i is not visited nor the leader, then it becomes visited, it informs its neighbors and it sends to its parent the token
$$< ID, m + 1, \Delta + 1, \Delta_i, \overline{\delta}_T >;$$
II) if v_i is visited and it is not the leader, it sends to its parent the token
$$< ID, m, \Delta^\dagger, \Delta_i, \overline{\delta}_T >$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

(recall that $\Delta^\dagger$ is the biggest depth among those of the nodes that branch from v_i along the tree);

III) if v_i is the leader, then it calculates the diameter of the tree as $\delta = \overline{\delta}_T$, the number m of nodes visited, and terminates the procedure.

In all the above cases, a node v_i with $\mathcal{M}_i = \emptyset$ is no more involved in the calculation of that particular tree, while it is involved in the calculation of successive spanning trees.

B. Discussion and Properties

Let us provide some results on the correctness of the above algorithm.

Theorem 3: Suppose G is undirected and connected. The DSTC Algorithm is correct, that is, at its completion

a) it finds a spanning tree over G;
b) the leader node knows the number n of nodes in G;
c) the leader node knows the diameter δ of the spanning tree.

Proof: The proof of point a) is trivial; since we execute a depth-first search of the nodes, we obtain a spanning tree. As for point b), the estimate m of the number of nodes is increased each time a not visited node receives the token, and the depth-first visit approach guarantees all nodes are visited.

Let us now prove point c). Note that, when a leaf node transmits the token to its parent, it transmits its depth over the tree, while non-leaf nodes v_i transmit to their parents the value of the maximum depth $\Delta^\dagger$ of one of the nodes in the branch of v_i. As a result, each node knows the correct value of $\Delta^\dagger$ and $\Delta^\ddagger$.

Let us consider the generic node v_i sending the token to its parent along the tree. If v_i is a leaf, then it sends $\delta_T^\dagger = \Delta_i$ to its parent. If it is not a leaf, nor the leader, it calculates $\overline{\delta}_T$ as the maximum between $\delta_T^\dagger$ and $\Delta^\dagger + \Delta^\ddagger - 2\Delta_i$. The first term is the maximum length of a path from the root node to one of the leaves in the branch of v_i. As for the second term, we claim that it is the maximum length of a path connecting the 2 farthest leaves (if there is more than one) in the branch of node v_i; in fact, the distance between v_i and its farthest leaf is $\Delta^\dagger - \Delta_i$, while the distance from the second farthest one (if any) is $\Delta^\ddagger - \Delta_i$.

Therefore, it can be noted that, when a node v_i transmits a token to its parent, all possible paths of maximum length involving v_i and the nodes in the branch of v_i have been inspected, and the maximum among the lengths of such paths is stored in $\overline{\delta}_T$; hence, when the last token is received by the root node it must hold $\overline{\delta}_T = \delta$. ■

Let us now discuss the computational characteristics of the proposed DSTC algorithm.

Proposition 1: The DSTC algorithm is such that:

a) it requires $2e + 2n - 2$ total messages and at most $2|\mathcal{N}_i|$ messages for each node;
b) each message is $O(\log n)$ bits;
c) the total memory required at each node is $O(|\mathcal{N}_i| \log n)$ bits.

Proof: a) If G is connected, then exactly $2n-2$ tokens are sent, because a token is sent twice along all the links assigned to the spanning tree, which are $n-1$. Since eventually all nodes are visited, each node provides an additional message to all its neighbors, and for each edge two such messages are sent, hence the total number of messages sent (without counting the leader election) is $2e + 2n - 2$, while each node sends at most $2|\mathcal{N}_i|$ messages (i.e., a token and a visited message to all its neighbors).

b) Each message requires $O(\log n)$ bits, as the quantities exchanged are Boolean variables or integers in the range $[0, n]$.

c) As for the memory requirements for each agent, they need to store a label for each neighbor, and using progressive integers for the ID of the trees each label requires $O(\log n)$ bits, hence the memory requirement is $O(|\mathcal{N}_i| \log n)$ bits for each agent. ■

Note that, the DSTC algorithm can be implemented in an asynchronous fashion, as only one message is sent at a time in the network. Notice further that, at the end of the procedure, the leader node knows the number n of nodes in the network and the diameter δ of the tree; this information can be provided to all other nodes by means of $n-1$ more messages (i.e., along the spanning tree).

In the next section we provide a methodology to solve Problem 1 in an approximated manner.

VI. Construction of Multiple EDSTs

As discussed above, in order to provide a suboptimal solution to Problem 1, we attempt to build multiple edge-disjoint spanning trees by iteratively executing the DSTC Algorithm and removing the edges in the found spanning tree, and so on until DSTC fails to construct a spanning tree; we refer to this algorithm as *Distributed Edge-Disjoint Spanning Trees Construction Algorithm* (DESTC), and we report its pseudocode in Algorithm 2.

Specifically, within DESTC the leader node starts a DSTC procedure, and when the procedure terminates each node discards the labeled edges. After the t-th run of DSTC, the leader starts a new DSTC if $t = 1$ (i.e., if only one DSTC has been performed) or if the number of visited nodes m_t during the t-th DSCT is equal to the number of nodes visited at the previous run. When the condition is not met, the leader stops and the algorithm terminates.

Note that, over a connected graph, there is the guarantee to find at least one spanning tree, hence the leader node is able to detect the failure or success of subsequent executions of the DSTC Algorithm by comparing the estimate of the number of nodes obtained at each execution after the first estimate.

Note further that there is no need to elect a new leader at each execution of the DSTC Algorithm; it is, however, possible to further reduce the overall computations by selecting different leaders, as discussed in the next remark.

Remark 6: If the number of nodes does not change during the execution of the DESTC Algorithm, a simple improvement

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

consists in executing once the DSTC algorithm, and then simplifying the successive runs of DSTC. Specifically, when the n-th node is visited, there is no need to traverse the links in the spanning tree up to the leader. It is sufficient to let the n-th node become the new leader and begin a new depth-first visit. Such an approach would result in a reduction of half of the time steps, and would be more robust because it does not rely on a single leader. However, for simplicity, in the following we neglect this possibility.

Algorithm 2 Distributed Edge-Disjoint Spanning Trees Construction (DESTC)

```
1:  procedure DESTC(is-leader)
2:      if is-leader then
3:          t ← 0;
4:          start first DSTC;
5:  procedure ONFINISHEDDSTC(is-leader)
6:      remove labeled edges from consideration;
7:      if is-leader then
8:          t ← t + 1;
9:          m_t ← visited nodes in t-th run of DSTC;
10:         if t = 1 or m_t = m_{t−1} then
11:             start new DSTC;
12:         else // DSTC failed to find a spanning tree
13:             stop;
```

A. Computational Properties

The following proposition characterizes the number of messages exchanged during the execution of the DESTC algorithm.

***Proposition** 2:* The overall number ϕ of messages exchanged within the DESTC Algorithm is such that

$$\phi \in \Big[k\psi, (k+1)\psi\Big),$$

where

$$\psi = 2e + 2(n-1)\Big[1 - \frac{(k-1)(k-2)}{2}\Big]$$

and k is the number of EDSTs found by Algorithm DESTC. Moreover, each node sends at most $2(k+1)|\mathcal{N}_i|$ messages.

Proof: The number of messages required for the first execution is $2e + 2(n-1)$, while for each execution t we need to discard

$$2\sum_{t=1}^{k}(t-1)(n-1)$$

links, because at each execution $n-1$ links are removed from the graph. Hence, the total number of messages required for k executions is

$$\phi' = k[2e + 2(n-1)] - 2k\sum_{t=1}^{k}(t-1)(n-1)$$

or, after some algebra, $\phi' = k\psi$. The root node may have some links left after k executions, hence the agents attempt to execute the DSTC Algorithm once more; the total number of messages exchanged is, therefore, $\phi < (k+1)\psi$.

As for the number of messages exchanged by each node, it is at most $k+1$ times the amount of messages exchanged within DSTC; hence, it is at most $2(k+1)|\mathcal{N}_i|$. ■

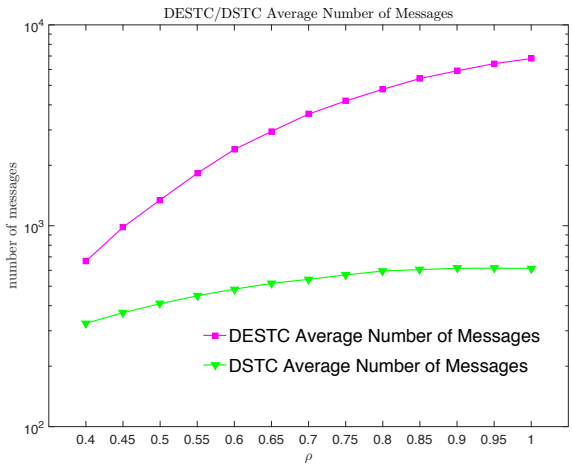

Fig. 4. Average number of messages for the DESTC algorithm (magenta squares) and for the DSTC algorithm (green triangles). As discussed in Section VII-A, we consider random geometric graphs with $n = 30$ and different values of ρ. The plot shows the average of the results for $m = 500$ trials.

***Remark** 7:* The memory requirements and the size of the messages within DESTC Algorithm coincide with those of DSTC Algorithm.

VII. Simulations

In this section we analyze the performance of the proposed DSTC and DESTC algorithms and we provide examples of applications that cope with MITM attacks.

A. Experimental Setting

In the following, we consider a *random geometric graph*, i.e., we place the nodes in uniformly distributed random positions in the unit square $[0,1]\times[0,1]$ and we connect them by an edge when their Euclidean distance is less than or equal to a threshold ρ.

B. DSTC and DESTC: Computational Properties

Figure 4 shows the average of the total number of messages exchanged by the agents during the DESTC algorithm (purple squares) and during the DSTC algorithm (green triangles facing down); it can be noted that the number of links in the graph (and the number of EDSTs found by DESTC Algorithm) grows as ρ grows, and the amount of messages exchanged grows accordingly.

C. DESTC versus Optimal but Centralized Approach

We now evaluate the results of DESTC in terms of the number of found EDSTs, depending on the communication radius ρ. Specifically, we compare the number of EDSTs found by the proposed algorithm (DESTC) against the algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

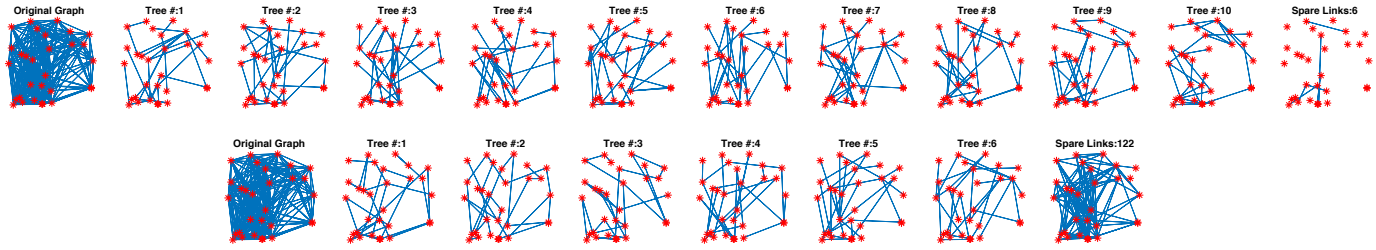

Fig. 5. Comparison between the algorithm from Roskind and Tarjan [30] (upper plot) and the DESTC Algorithm (lower plot), with respect to the same random geometric graph with $n = 30$ and $\rho = 0.65$.

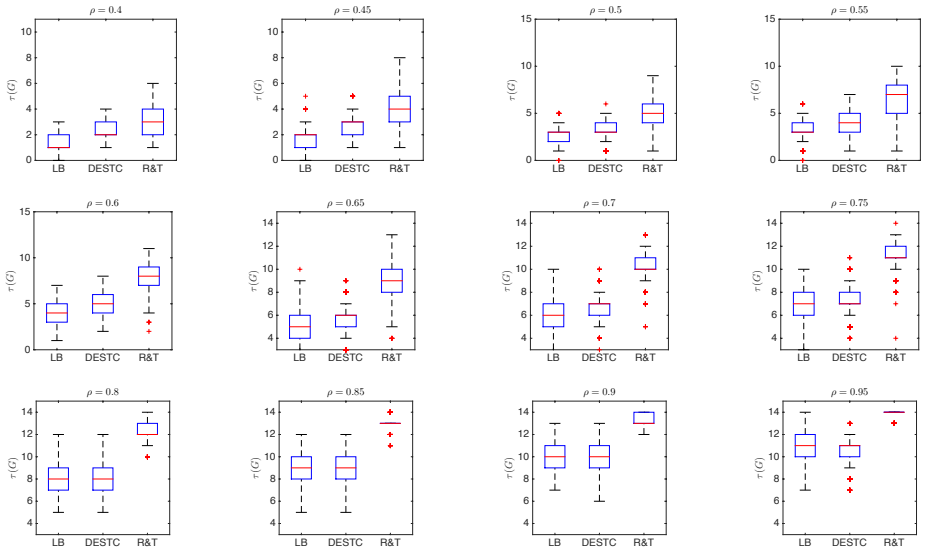

Fig. 6. Comparison of the number of EDSTs found by the proposed algorithm (DESTC) against the algorithm from Roskind and Tarjan (R&T) [30] and against the lower bound (LB) discussed in Section IV-C, considering random geometric graphs with $n = 30$ nodes and different choices of ρ. Each plot shows the distribution of the results for $m = 500$ trials.

from Roskind and Tarjan (R&T) [30] and against the lower bound (LB) discussed in Section IV-C.

Figure 5 shows a comparison of DESTC (lower plots) and [30] R&T (upper plots) on a particular instance with $n = 30$ nodes and $\rho = 0.65$. As shown by the picture, DESTC Algorithm finds 6 EDSTs, while the algorithm in [30] finds $\tau(G) = 10$ EDSTs. Within DESTC Algorithm, instead, there are 122 spare links that constitute a disconnected graph, implying that a different choice of the links would have yielded a bigger number of EDSTs.

In Figure 6, we consider random geometric graphs with $n = 30$ nodes and for each choice of $\rho \in [0.4, 0.95]$ we generate 500 connected graphs. According to the figure, we note that the median value for DESTC is above the one for LB when $\rho \leq 0.8$; moreover, it can be noted that the 5^{th} percentile for DESTC is above the median of LB for $\rho = 0.4$, and it is above the 25^{th} percentile of LB for $\rho = 0.45$ For higher values of ρ, the results for DESTC and LB tend to coincide. As for the comparison with R&T, it can be noted that for $\rho < 0.5$ the results are comparable for the median values, although R&T has better results in terms of 75^{th} and 95^{th} percentile. For $\rho \geq 0.5$, instead, the 75^{th} percentile for DESTC is below the 25^{th} percentile for R&T, while for $\rho \geq 0.75$ the 75^{th} percentile for DESTC is below the 5^{th} percentile for R&T. The results suggest that the proposed algorithm has good performance for relatively sparse networks, while it gets increasingly closer to the lower bound as ρ grows; it should be noted, however, that the distributed setting is of particular interest in the case of sparse networks, therefore the proposed algorithm appears well justified. Note that, in the above simulation, the EDSTs found by DESTC have average diameter between 24 ($\rho = 0.4$) and 25.67 ($\rho = 0.95$), while R&T obtains considerably smaller average diameter, being between 8 and 10; in both cases, however, the average diameter is well above the diameter of the graph being partitioned, which in our trials is around 4 for $\rho = 0.4$ up to 2 for $\rho = 0.95$.

D. Coping with MITM Attacks

We now provide an example showing the potential of the proposed approach in terms of detection of MITM attacks and restoration of the correct result. We consider a random

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

geometric graph with $n = 30$ nodes and $\rho = 0.7$, resulting in $|E| = 363$ links. By means of DESTC Algorithm, we find 9 EDSTs and we analyze the effectiveness of MITM attacks affecting distributed algorithms running in parallel over the found EDSTs.

Specifically, in Figure 7 we report the results obtained with respect to agents executing the max-consensus algorithm (upper plot, see [45] and references therein for details on max-consensus algorithms) and the average-consensus algorithm [1] (lower plot). Note that, since the average-consensus algorithm has asymptotic convergence, we implement a stopping criterion and we approximate the result of each instance. In each plot we consider MITM attacks affecting a given number of links (up to 30) selected at random and we plot, depending on the number of attacked links the percentage of successful detections of the attack (in red) and the percentage of successful restoration of the correct result (in blue). In the max-consensus case we assume the i-th agent starts with initial condition equal to i (so that the maximum is $n = 30$) and the attackers replace the transmitted values with a value $2n = 60$, so that by affecting one link the result of the corresponding EDSSG is 60 instead of 30. In the average-consensus case we do the same (hence the average is 15.5), but since we approximate the asymptotic result at a given point, we assume that solutions differing of less than 0.1 are counted as if they were the same result. According to the figure, the ability to cope with MITM attacks is radically different in the case of max-consensus and average-consensus. In the first case the agents are able to detect the attack with high probability in spite of the attacks; the detection percentage is around 100% if up to 13 links are attacked, while it degrades as more links are attacked (for 30 links attacked the detection percentage is around 20%). In the case of average-consensus, instead, the percentage is always 100%, since the attackers have no obvious way to guarantee that each attacked instance will yield the same result, hence in general the attacks yield different results, and the agents are able to spot the attack. As for the restoration probability, according to Remark 2, having 9 EDSTs we are guaranteed to be able to restore attacks to no more than 4 links in all cases, while the percentage degrades as more links are attacked. However, the results for average-consensus are remarkably better than max-consensus; as discussed above, the attackers might not be able to steer the result of the different instances to the same result; hence, although not being the absolute majority, there might be a relative majority of instances that are not attacked (e.g., two instances having the same value while all other instances have erroneous but different values).

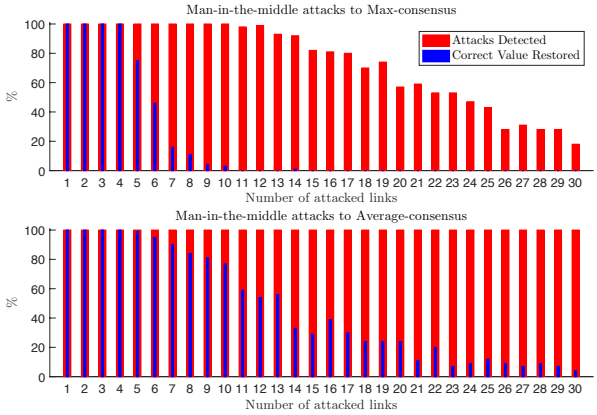

Fig. 7. Example of detection of MITM attacks and restoration of the correct result in the case of max-consensus (upper plot) and average consensus (lower plot), with respect to a random geometric graph with $n = 30$ nodes and $\rho = 0.7$. In this case, DESTC algorithm finds 9 EDSTSs and run the algorithms in parallel over the EDSTs. We plot the results in terms of detection percentage (in red) and restoration percentage (in blue) when different number of links are attacked at random. We consider 100 trials for each number of attacked links.

VIII. Conclusions

In this paper we provide a lightweight and asynchronous distributed methodology to construct a set of EDSTs. Such EDSTs can be used to execute in parallel several instances of the same distributed algorithm, so that the algorithm becomes robust to man-in-the-middle attacks. Future work will be aimed at extending the methodology to directed graphs and to increase the degree of parallelism in the construction of the EDSTs. Moreover we will investigate strategies to construct a smaller set of EDSSGs with small diameter; to this end the ability of DSCT Algorithm to find the diameter of the EDSTs it constructs will be highly beneficial. Another important research direction is the definition of performance bounds of the proposed approach with respect to the optimal but centralized algorithm.

References

[1] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[2] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[3] H. Moniz, N. F. Neves, and M. Correia, "Byzantine fault-tolerant consensus in wireless ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 12, pp. 2441–2454, 2013.

[4] L. Tseng and N. H. Vaidya, "Fault-tolerant consensus in directed graphs," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. ACM, 2015, pp. 451–460.

[5] H. Chen, "Decentralized consensus for P2P network with trust relationships," *arXiv preprint arXiv:1501.06238*, 2015.

[6] M. Cai, Z. Xiang, and J. Guo, "Adaptive finite-time fault-tolerant consensus protocols for multiple mechanical systems," *Journal of the Franklin Institute*, vol. 353, no. 6, pp. 1386–1408, 2016.

[7] G. Chen and Y.-D. Song, "Robust fault-tolerant cooperative control of multi-agent systems: A constructive design method," *Journal of the Franklin Institute*, vol. 352, no. 10, pp. 4045–4066, 2015.

[8] Y. Ge, J. Wang, L. Zhang, B. Wang, and C. Li, "Robust fault tolerant control of distributed networked control systems with variable structure," *Journal of the Franklin Institute*, vol. 353, no. 12, pp. 2553–2575, 2016.

[9] F. Fagnani and S. Zampieri. "Average consensus with packet drop communication", *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 102–133, 2009.

[10] C. Kai and H. Ishii, "Average consensus on arbitrary strongly connected digraphs with time-varying topologies", *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 1066-1071, 2014.

[11] C. N. Hadjicostis, H. V. Nitin and A. D. Domínguez-García,"Robust distributed average consensus via exchange of running sums", *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1492–1507, 2016.

[12] M. Strebe and C. L. Perkins, *Firewalls: 24 Seven*. Sybex Inc., 1999.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

[13] Y. Desmedt, "Man-in-the-middle attack," in *Encyclopedia of cryptography and security*. Springer, 2011, pp. 759–759.
[14] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.
[15] J. T. Chiang, J. J. Haas, Y.-C. Hu, P. Kumar, and J. Choi, "Fundamental limits on secure clock synchronization and man-in-the-middle detection in fixed wireless networks," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 1962–1970.
[16] M. Rahman and K. El-Khatib, "Secure time synchronization for wireless sensor networks based on bilinear pairing functions," *IEEE Transactions on Parallel and Distributed Systems*, 2010.
[17] L. Wang and A. M. Wyglinski, "Detection of man-in-the-middle attacks using physical layer wireless security techniques," *Wireless Communications and Mobile Computing*, 2014.
[18] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Information and Computation*, vol. 79, no. 1, pp. 43–59, 1988.
[19] Y. Challal, A. Ouadjaout, N. Lasla, M. Bagaa, and A. Hadjidj, "Secure and efficient disjoint multipath construction for fault tolerant routing in wireless sensor networks," *Journal of network and computer applications*, vol. 34, no. 4, pp. 1380–1397, 2011.
[20] S. Chakraborty, S. Chakraborty, S. Nandi, and S. Karmakar, "Fault resilience in sensor networks: distributed node-disjoint multi-path multi-sink forwarding," *Journal of Network and Computer Applications*, vol. 57, pp. 85–101, 2015.
[21] S. Sundaram and C. N. Hadjicostis, "Distributed function calculation via linear iterative strategies in the presence of malicious agents", *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1495–1508, 2011.
[22] Y.-C. Tseng, S.-Y. Wang, and C.-W. Ho, "Efficient broadcasting in wormhole-routed multicomputers: A network-partitioning approach," *IEEE Transactions on Parallel and Distributed systems*, vol. 10, no. 1, pp. 44–61, 1999.
[23] O. Beaumont, N. Bonichon, L. Eyraud-Dubois, P. Uznański, and S. K. Agrawal, "Broadcasting on large scale heterogeneous platforms under the bounded multi-port model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2520–2528, 2014.
[24] W. T. Tutte, "On the problem of decomposing a graph into n connected factors," *Journal of the London Mathematical Society*, vol. 1, no. 1, pp. 221–230, 1961.
[25] C. S. J. Nash-Williams, "Edge-disjoint spanning trees of finite graphs," *Journal of the London Mathematical Society*, vol. 1, no. 1, pp. 445–450, 1961.
[26] T. Kaiser, "A short proof of the tree-packing theorem," *Discrete Mathematics*, vol. 312, no. 10, pp. 1689–1691, 2012.
[27] S. M. Cioabă and W. Wong, "Edge-disjoint spanning trees and eigenvalues of regular graphs," *Linear Algebra and its Applications*, vol. 437, no. 2, pp. 630–647, 2012.
[28] Q. Liu, Y. Hong, X. Gu, and H.-J. Lai, "Note on edge-disjoint spanning trees and eigenvalues," *Linear Algebra and its Applications*, vol. 458, pp. 128–133, 2014.
[29] A. Frieze and T. Johansson, "On edge disjoint spanning trees in a randomly weighted complete graph," *arXiv preprint arXiv:1505.03429*, 2015.
[30] J. Roskind and R. E. Tarjan, "A note on finding minimum-cost edge-disjoint spanning trees," *Mathematics of Operations Research*, vol. 10, no. 4, pp. 701–708, 1985.
[31] J. Edmonds, "Matroids and the greedy algorithm," *Mathematical programming*, vol. 1, no. 1, pp. 127–136, 1971.
[32] Y. Wang, J. Fan, X. Jia, and H. Huang, "An algorithm to construct independent spanning trees on parity cubes," *Theoretical Computer Science*, vol. 465, pp. 61–72, 2012.
[33] Y. Wang, J. Fan, W. Liu, and Y. Han, "A parallel algorithm to construct BISTs on parity cubes," in *Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on*. IEEE, 2015, pp. 54–58.
[34] V. Yadav and M. V. Salapaka, "Distributed protocol for determining when averaging consensus is reached", *Annual Allerton Conference*, pp. 715–720, 2007.
[35] N. E. Manitara and C. N. Hadjicostis, "Distributed stopping for average consensus in undirected graphs via event-triggered strategies," *Automatica*, vol. 70, pp. 121–127, 2016.
[36] T. Charalambous, Y. Yuan, T. Yang, W. Pan, C. N. Hadjicostis and M. Johansson, "Distributed finite-time average consensus in digraphs in the presence of time delays", *IEEE Transactions on Control of Network Systems*, vol. 2, no. 4, pp. 370–381, 2015.
[37] L. Lovász, *Combinatorial problems and exercises*. American Mathematical Soc., 1993, vol. 361.
[38] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 215–225, 1975.
[39] W. Wong, *Spanning trees, toughness, and eigenvalues of regular graphs*. University of Delaware, 2013.
[40] B. Awerbuch, "A new distributed depth-first-search algorithm," *Information Processing Letters*, vol. 20, no. 3, pp. 147–150, 1985.
[41] K. Lakshmanan, N. Meenakshi, and K. Thulasiraman, "A time-optimal message-efficient distributed algorithm for depth-first-search," *Information Processing Letters*, vol. 25, no. 2, pp. 103–109, 1987.
[42] S. Makki and G. Havas, "Distributed algorithms for depth-first search," *Information Processing Letters*, vol. 60, no. 1, pp. 7–12, 1996.
[43] M. Raynal, *Distributed algorithms for message-passing systems*. Springer, 2013, vol. 500.
[44] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan, "Sublinear bounds for randomized leader election," *Theoretical Computer Science*, vol. 561, pp. 134–143, 2015.
[45] F. Iutzeler, P. Ciblat, and J. Jakubowicz, "Analysis of max-consensus algorithms in wireless channels," *IEEE Transactions on Signal Processing*, vol. 60, no. 11, pp. 6103–6107, 2012.

Gabriele Oliva (M'11) received the Laurea degree and the Ph.D in Computer Science and Automation Engineering in 2008 and 2012, respectively, both at University Roma Tre of Rome, Italy. He is currently assistant professor in Automatic Control at the University Campus Bio-Medico of Rome, Italy. His main research interests include distributed systems, distributed optimization, and applications of graph theory in technological and biological systems.

Sebastian Cioabă received the B.Sc degree in Mathematics and Computer Science in 2000 at the University of Bucharest, Romania, the M.Sc and Ph.D degrees in Mathematics in 2002 and 2005, respectively, both at the the Queen's University of Kingston, Ontario. After postdoctoral positions at University of California, San Diego and University of Toronto, he has been at University of Delaware since 2009. He currently serves as Associate Professor in the Department of Mathematical Sciences at University of Delaware in Newark, Delaware. His main research interests lie in graph theory, combinatorics and their applications to other areas of mathematics and science. He authored or co-authored more than 40 papers published in peer-reviewed international journals. He is on the editorial board of Linear and Multilinear Algebra and Electronic Journal of Linear Algebra.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCNS.2017.2746344, IEEE Transactions on Control of Network Systems

Christoforos N. Hadjicostis (M'99, SM'05) received the S.B. degrees in electrical engineering, computer science and engineering, and in mathematics, the M.Eng. degree in electrical engineering and computer science in 1995, and the Ph.D. degree in electrical engineering and computer science in 1999, all from Massachusetts Institute of Technology, Cambridge. In 1999, he joined the Faculty at the University of Illinois at Urbana–Champaign, where he served as Assistant and then Associate Professor with the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Information Trust Institute. Since 2007, he has been with the Department of Electrical and Computer Engineering, University of Cyprus, where he is currently Professor and Dean of Engineering. His research focuses on fault diagnosis and tolerance in distributed dynamic systems, error control coding, monitoring, diagnosis and control of large-scale discrete-event systems, and applications to network security, anomaly detection, energy distribution systems, medical diagnosis, biosequencing, and genetic regulatory models. He currently serves as Associate Editor of IEEE Transactions on Automatic Control, and IEEE Transactions on Automation Science and Engineering; he has also served as Associate Editor of IEEE Transactions on Control Systems Technology, and IEEE Transactions on Circuits and Systems I.