Identification of Malicious Activity in Distributed Average Consensus via Non-Concurrent Checking

Christoforos N. Hadjicostis and Alejandro D. Domínguez-García

***Abstract*— We consider the problem of average consensus in a network system under a fixed, undirected communication topology, when there are malicious nodes present that may try to influence the average calculation. In the setting considered, the average consensus is performed by the nodes in a distributed fashion using a linear iterative algorithm. We assume malicious nodes can manipulate, in an arbitrary manner, the value of their state in the aforementioned algorithm; the problem is then to check whether or not each node is correctly performing the updates of its state. To address this problem, we propose a distributed algorithm whereby each node is in charge of checking the updates performed by its neighboring nodes based on information that it receives from them and also from the neighbors of its neighbors. The algorithm leverages ideas from non-concurrent error detection schemes and its main advantage is that information from two-hop neighbors is only needed infrequently—a relaxation that significantly reduces the communication overhead associated with the requirement to make such information available.**

***Index Terms*— Agents-based systems, Distributed control, Fault detection, Network analysis and control**

I. Introduction and Motivation

We consider network systems consisting of a set of nodes that can share information with neighboring nodes via connection links, forming an interconnection topology, which can be described by a graph. In such systems, it is often necessary for all or some of the nodes to calculate a function of certain parameters, referred to as initial values, held locally by the nodes [1], [2]. For example, when all nodes calculate the average of these initial values, they are said to reach average consensus. Due to its usage in numerous distributed control, computing, and communication applications, distributed algorithms for average consensus have been researched extensively (see, e.g., [1]–[5]). An important class of such algorithms rely on a linear iteration whose convergence is asymptotic (see, e.g., [1], [2], [4] and the references therein). This paper focuses on the problem of detecting/identifying malicious nodes in a network system during the execution of such average consensus algorithms. In our setting, malicious nodes are loosely defined as nodes that are performing incorrect updates in an attempt to manipulate the outcome of the average consensus algorithm.

To address the problem, we propose a linear iterative scheme that allows each node of a network system to check, in a distributed fashion, whether or not its neighboring nodes have performed their updates correctly. The proposed distributed scheme, which is based on the centralized non-concurrent error detection and identification scheme in [6], relies on additional (internal) state variables that are maintained at each node for monitoring purposes. The update performed by each node of said variables uses information from two-hop neighbors only periodically (instead of requiring such information at each iteration). Note that, if such *two-hop* information is available at each iteration, then the checker node can essentially emulate the computations that each of its neighboring nodes under check is supposed to perform, and verify (separately, for each of its neighbors) whether its updates are correct; this is essentially the approach taken in [7]–[9].

The main contributions of the paper are as follows. In comparison with [6], which focused exclusively in sparse errors, i.e., transient faults injected by some nodes at specific time steps, the errors injected by malicious nodes in our setting are not necessarily sparse, i.e., a malicious node may be introducing errors in its value at every iteration. Then, unlike the algorithm proposed in [6], our algorithm is able to detect such non-sparse errors as long as they accumulate over time to a nonzero value between the periodic checks performed by the algorithm. By contrast with the schemes proposed in [7]–[9], the non-concurrent checking scheme we propose in this paper does not require each node to maintain a different monitor for each of its neighbors and, perhaps more importantly, it requires information from the two-hop neighbors only once every few iterations; thus, significantly reducing communication requirements.

In the paper we will not elaborate on ways to handle malicious nodes once they are declared as such (or when they try to incriminate other nodes by providing incorrect information or assessments) because one can proceed using a variety of existing techniques. For example, one can rely on a separate mechanism to inform non-malicious nodes about which node(s) has (have) been declared malicious, and non-

C. N. Hadjicostis is with the ECE Department at the University of Cyprus, Nicosia, Cyprus, and also with the ECE Department at the University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. E-mail: hadjicostis.christoforos@ucy.ac.cy.

Alejandro D. Domínguez-García is with the ECE Department at the University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. E-mail: aledan@illinois.edu.

The work of Domínguez-García has been funded in part by the U.S. Department of Defense's Environmental Security Technology Certification Program (ESTCP) under Contract Number W912HQ-20-C-0039.

malicious nodes can continue their iteration while ignoring the malicious nodes (see, [7]–[10]). In addition, one could use the approach in [11] to ensure that nodes completely remove the effects of malicious nodes on the computation of the average. Alternatively, one could use the approach proposed in [12] to limit the effect of malicious nodes on the computation; however, such approach would only guarantee reaching consensus to a value between the minimum and maximum value of non-malicious nodes.

II. Mathematical Background and Notation

A. Graph-Theoretic Notions and Communication Model

A directed graph (digraph) of order N ($N \geq 2$), is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} - \{(v_j, v_j) \mid v_j \in \mathcal{V}\}$ is the set of edges. A directed edge from node v_i to node v_j is denoted by $(v_j, v_i) \in \mathcal{E}$, and indicates that node v_i can send information to node v_j. A digraph is called undirected, denoted $\mathcal{G}_u = (\mathcal{V}, \mathcal{E})$, if $(v_j, v_i) \in \mathcal{E}$ implies that $(v_i, v_j) \in \mathcal{E}$. In such case, the set of nodes that can send/receive information directly to/from node v_j, which we denote by $\mathcal{N}_j$, is referred to as the neighborhood of v_j, i.e., $\mathcal{N}_j = \{v_i \in \mathcal{V} \mid (v_j, v_i) \in \mathcal{E}\} = \{v_l \in \mathcal{V} \mid (v_l, v_j) \in \mathcal{E}\}$. Each element of $\mathcal{N}_j$ is referred to as a neighbor of v_j and the cardinality of $\mathcal{N}_j$, which we denote by D_j, is referred to as the degree of node v_j. We say that $\mathcal{G}_u$ is *connected* if for each pair of nodes $v_j, v_i \in \mathcal{V}$, $v_j \neq v_i$, there exists a path from v_i to v_j, i.e., we can find a sequence of nodes $v_i =: v_{l_0}, v_{l_1}, \ldots, v_{l_t} := v_j$ such that $(v_{l_{\tau+1}}, v_{l_\tau})$ (thus, necessarily $(v_{l_\tau}, v_{l_{\tau+1}})$) belongs in $\mathcal{E}$ for $\tau = 0, 1, \ldots, t-1$.

B. Distributed Average Consensus in Undirected Graphs

Consider a network system, captured by an undirected graph $\mathcal{G}_u = (\mathcal{V}, \mathcal{E})$, in which each node $v_j \in \mathcal{V}$ has an initial value V_j. Average consensus aims to have all the nodes calculate the average of their initial values, denoted by $\overline{V} = \frac{\sum_{\ell=1}^{N} V_\ell}{N}$. The algorithm we consider in this section is iterative. Specifically, each node v_j maintains a scalar state variable $x_j[k]$ and updates it at each iteration k as follows:

$$x_j[k+1] = p_{jj} x_j[k] + \sum_{v_i \in \mathcal{N}_j} p_{ji} x_i[k] \ , \qquad (1)$$

where $x_j[0] = V_j$, $v_j \in \mathcal{V}$, and the p_{ji}'s are constant weights. If we let $x[k] = [x_1[k], x_2[k], \ldots, x_N[k]]^{\mathrm{T}}$, then the iteration in (1) can be written in matrix form as

$$x[k+1] = P x[k] \ , \ \ x[0] = [V_1, V_2, \ldots, V_N]^{\mathrm{T}}, \qquad (2)$$

where $P = [p_{ji}]$ is the weight matrix with the constraint that the entry p_{ji} at its jth row and ith column satisfies $p_{ji} = 0$ if $v_i \notin \mathcal{N}_j \cup \{v_j\}$. The nodes are said to reach asymptotic average consensus if $\lim_{k \to \infty} x_j[k] = \overline{V}$ for all $v_j \in \mathcal{V}$.

As stated in [4], [13], the necessary and sufficient conditions for the iteration in (2) to asymptotically reach average consensus are: (i) P has a simple eigenvalue at 1, with left eigenvector 1_N^T and right eigenvector 1_N (where 1_N denotes the all ones column vector with N entries), and (ii) all other eigenvalues of P have magnitude strictly less than 1. If one focuses on nonnegative weights, these conditions are equivalent to P being a primitive doubly stochastic matrix. In the case of an undirected graph, there are very simple ways for the nodes to choose the weights in a distributed manner so that P forms a primitive doubly stochastic matrix [2]. For example, assuming the nodes know the total number of nodes N or an upper bound $N' \geq N$, each node v_j sets the weights on all of its incoming links to be $p_{ji} = \frac{1}{N'}$ for all $v_i \in \mathcal{N}_j$ and $p_{jj} = 1 - \frac{D_j}{N'}$ for $i = j$ (zero otherwise). It is easy to verify that P will be a symmetric doubly stochastic matrix; P will be primitive as long as $\mathcal{G}_u$ is connected.

III. Non-Concurrent Checking for Identifying Faulty/Malicious Nodes

In this section, we propose a scheme for non-concurrent error checking in the distributed average consensus algorithm in (1). The idea is based on having each node v_j check for errors introduced by other (faulty or malicious) nodes in its neighborhood $\mathcal{N}_j$. To this end, node v_j needs to maintain some additional (internal) state variables, which it updates based on information that it receives directly from these nodes at each iteration, as well as information that it receives periodically, namely once every K iterations, from its two-hop neighbors, i.e., the neighbors of its neighbors. Node v_j also needs to maintain one additional variable to send to all of its two-hop neighbors every K iterations so that these nodes can perform their checks on their neighbors.

Definition 1: A node v_j that participates in the average consensus computation in (1) is called malicious if during at least one iteration k, the update of its value $x_j[k+1]$ is incorrect, i.e., for some iteration k, we have $x_j[k+1] = p_{jj} x_j[k] + \sum_{v_i \in \mathcal{N}_j} p_{ji} x_i[k] + e_j[k]$ for a nonzero $e_j[k]$.

A. Model of Subgraph to be Checked by Node v_j

In the remainder we assume the following holds.

Assumption 1. Consider a connected undirected graph $\mathcal{G}_u = (\mathcal{V}, \mathcal{E})$ where nodes are executing a linear update of the form in (1) to reach average consensus. Each node v_j is aware of the local topology around it up to two hops, i.e., it is aware of its neighbors and the neighbors of its neighbors (i.e., the set $\mathcal{N}_j^{(2)} := \mathcal{N}_j \cup (\cup_{v_i \in \mathcal{N}_j} \mathcal{N}_i)$), as well as their interconnections and weights used by them in the update in (1) (i.e., the set of weights $\{p_{li} \mid v_l, v_i \in \mathcal{N}_j^{(2)} \cup \{v_j\}\}$).

Assumption 2. Each node v_j is capable of two types of transmissions, namely, (i) broadcasting messages that are received by all of its neighbors $\mathcal{N}_j$, and (ii) two-hop broadcasting messages that are received by all of its neighbors and all the neighbors of its neighbors (i.e., all nodes in its two-hop neighborhood $\mathcal{N}_j^{(2)}$). Furthermore, we assume that both types of transmissions are associated with a unique node ID that allows receiving nodes to identify the sending node.

Remark 1: Messages to two-hop neighbors can be sent in various ways, e.g., by transmitting at a higher power. This is likely to be more expensive and undesirable and is one of the reasons we propose a scheme that uses such transmissions less frequently (once every K iterations). The ID of the node that sends a message is needed to be able to identify the correct weighting factor in the calculations and to properly declare which nodes are malicious.

We now provide a model describing the part of the consensus algorithm in (1) to be checked from errors by each node. For ease of notation and without loss of generality, we describe the model from the point of view of node v_1, which it is assumed to have $n-1$ neighbors, whose labels are such that $\mathcal{N}_1 = \{v_2, v_3, ..., v_n\}$. In addition, we assume that v_1 has $m \geq 1$ two-hop neighbors, whose labels are such that $\mathcal{N}_1^{(2)} \setminus (\mathcal{N}_1 \cup \{v_1\}) = \{v_{n+1}, v_{n+2}, ..., v_{n+m}\}$. [Since $\mathcal{G}_u$ is strongly connected under Assumption 1, if $m = 0$, it follows that node v_1 can directly communicate with all nodes which eases its task.]

With the above notation at hand, the first n entries of vector $x[k]$ in the iteration in (2) can be written as

$$q[k+1] = Aq[k] + Bu[k] \ , \tag{3}$$

where $q[k] = [x_1[k],\ x_2[k],\ ...,\ x_n[k]]^T$ is an n-dimensional vector, $u[k] = [x_{n+1}[k],\ x_{n+2}[k],\ ...,\ x_{n+m}[k]]^T$ is an m-dimensional vector (with $u_i[k] = x_{n+i}[k]$), A is an $n \times n$ matrix with entries $A(j,i) = P(j,i)$ for $1 \leq i, j \leq n$, and B is an $n \times m$ matrix with entries $B(j,i) = P(j, n+i)$ for $1 \leq j \leq n$ and $1 \leq i \leq m$. Effectively, the nodes in the set $\mathcal{N}_1 \cup \{v_1\}$ form a subnetwork and all other nodes that influence one or more nodes in this subnetwork (i.e., nodes in the set $\mathcal{N}_1^{(2)} \setminus (\mathcal{N}_1 \cup \{v_1\})$ are treated as exogenous inputs to this subnetwork. Note that the first row of matrix B is zero since the update of $x_1[k+1]$ only depends on the x variables of neighbors of node v_1 and node v_1 itself.

Next, we introduce the non-concurrent error detection scheme to be used by node v_1 to detect errors in (3) for the following two cases: i) two-hop information is received at each iteration k of the execution of (1), ii) two-hop information is received every $K > 1$ iterations.

B. Two-Hop Information Received at Each Iteration

In order to protect against additive errors that corrupt the value of $q[k]$, $k \geq 0$, we use the approach in [6] and construct a redundant version of the system in (3) of the form

$$r[k+1] = \mathcal{A}r[k] + \mathcal{B}u[k] \ , \tag{4}$$

where $r[k]$ is an $(n+2d)$-dimensional (redundant) state vector, and $\mathcal{A}$ and $\mathcal{B}$ are real matrices of appropriate dimensions to be chosen (d is a positive integer parameter that determines the number of errors that can be detected and identified, and will be discussed later). The system in (4) is designed so that it allows one to recover $q[k]$ from $r[k]$ (via some linear decoding mapping L to be determined), and to perform non-concurrent error detection and identification.

Consider that the redundant system operates in the interval $[0, K-1]$ (more generally, in any K-length window where K is a design parameter) and that during its operation within this interval, errors may occur. More specifically, an error $e[k]$ that occurs at time step k corrupts the state of the redundant system at instant $k+1$ as

$$r'[k+1] = \mathcal{A}r'[k] + \mathcal{B}u[k] + e[k] \ ,$$

where $r'[k]$ is used to distinguish the state of the redundant system at instant k in the presence of errors from the state at instant k, $r[k]$, that the system would be at in the absence of errors (note that $r[k]$, $k = 0, 1, 2, ..., K-1$, is generated by (4) under no errors in the time window $[0, K-1]$). It is not hard to verify that

$$r'[K] = r[K] + \sum_{t=0}^{K-1} \mathcal{A}^{K-1-t} e[t], \tag{5}$$

where $r[K]$ is the state the system would be in at the end of the time window if there were no errors, and $e[t]$, $t = 0, 1, ..., K-1$, are $(n+2d)$-dimensional error vectors.

In order to design our error detection scheme, we leverage the following particular construction of $\mathcal{A}$ and $\mathcal{B}$ from [6]:

$$\mathcal{A} = \left[\begin{array}{c|c} A & 0 \\ \hline CA - DC & D \end{array}\right] \quad , \quad \mathcal{B} = \left[\begin{array}{c} B \\ \hline CB \end{array}\right] \quad , \tag{6}$$

where C and D are real matrices of appropriate dimensions to be chosen. Note that with the above construction and in the absence of errors, we have that $q[k] = Lr[k]$, where L is an $n \times (n+2d)$ matrix of the form $L = [I_n\ 0]$, where I_n is the $n \times n$ identity matrix. Furthermore, one can use induction to show that, as long as $r[0] = \begin{bmatrix} q[0] \\ Cq[0] \end{bmatrix}$, the state $r[t]$ will satisfy $r[t] = \begin{bmatrix} q[t] \\ Cq[t] \end{bmatrix}$ for all t, regardless of the values of the input vector $u[t]$. This means that in the absence of errors, $Qr[t] = 0$ for all t, where $Q := [-C\ I_{2d}]$ is referred to as the *parity check* matrix. This also implies that if the syndrome $p[K] := Qr'[K]$ (which can be readily calculated at the end of the K-length interval) is nonzero, then one or more errors are detected; in fact, under proper choice of C, D, analysis of $p[K]$ can also lead to error identification [6].

Choice of Design Matrices C and D: In [6], the specific choices for C and D aimed to identify at most d' ($d' \leq d$) nonzero entries in the $((n+2d)K)$-dimensional vector

$$\overline{e} = [e^T[0],\ e^T[1],\ ...,\ e^T[K-1]]^T \ , \tag{7}$$

which captures all errors that have been injected during the K-length interval of the redundant system operation. In other words, [6] assumes that the vector $\overline{e}$ is sparse with d' nonzero entries (nonzero entry $\overline{e}_{(i_\ell + k_\ell(n+2d))} = \epsilon_\ell$ for $1 \leq \ell \leq d'$ captures an additive error of value ϵ_ℓ that corrupts the i_ℓth state variable of $r'[k]$ at iteration $K-1-k_\ell$). Note that i_ℓ is an index between 1 and $n+2d$, ϵ_ℓ is any real number, and k_ℓ is an integer between 0 and $K-1$. The choices for C and D were made in [6] to facilitate non-concurrent detection and identification of such sparse errors.

Here, we are interested in detecting and identifying errors that are injected by malicious nodes. Since a malicious node can inject an error at each time step of its operation, the error vector $\overline{e}$ in (7) is not necessarily sparse. In particular, if node v_i, $v_i \in \mathcal{N}_1$, is malicious then all of $\overline{e}_\ell$, $\ell = i + k(n+2d)$ for $k = 0, 1, \ldots, K-1$, can be nonzero. Towards this purpose, we set $D = I_{2d}$ in (6). With this choice, we can use induction to establish that, for $k = 1, 2, ...$, it holds

$$\mathcal{A}^k = \left[\begin{array}{c|c} A^k & 0 \\ \hline CA^k - C & I_{2d} \end{array}\right] \ .$$

Given the parity check matrix Q is of the form $Q = [-C \ \ I_{2d}]$, one can also establish that $Q\mathcal{A}^k = Q$ for all $k = 0, 1, 2, \ldots$. Since $Qr[K] = 0$, by using (5), we have that

$$p[K] = Qr'[K] = Q\sum_{t=0}^{K-1}\mathcal{A}^{K-1-t}e[t] = Q\sum_{t=0}^{K-1}e[t] \ . \quad (8)$$

Let $e_i[t]$, $i = 2, 3, ..., n$, be the scalar additive error introduced by node v_i at iteration t. Note that $e_i[t]$ is zero if no error is introduced by neighboring node v_i at iteration t. As we are describing checking from the point of view of node v_1, the error introduced by node v_1 can be safely assumed to be zero, i.e., for all $t = 0, 1, .., K-1$, we have $e_1[t] = 0$ and $e_{n+j}[t] = 0$ for $j = 1, 2, ..., m$ (recall that the bottom $2d$ variables are computed at node v_1). If we use $\varepsilon_i[K] = \sum_{t=0}^{K-1} e_i[t]$ to denote the total error introduced by node v_i, we can simplify (8) to $p[K] = [-C \ I_{2d}]\varepsilon[K]$, with

$$\varepsilon[K] = \begin{bmatrix} 0, \ \varepsilon_2[K], \ ...\varepsilon_n[K], \ \underbrace{0, 0, ..., 0}_{2d \text{ zeros}} \end{bmatrix}^T \ . \quad (9)$$

If we have at most d' nonzero $\varepsilon_i[K]$'s, the $p[K]$ will be a linear combination of d' columns of matrix $-C$. Therefore, rank conditions on the columns of C determine our ability to detect and identify malicious nodes based on the parity check $p[K]$. The following lemma summarizes the benefits of one such set of rank conditions.

Lemma 1: Suppose we build the monitor in (4)–(6) with matrix C having the property that any $2d$ columns of it are linearly independent. If each malicious node v_i is associated with a nonzero $\varepsilon_i[K]$ at time instant K in (9), then we are guaranteed (i) detection of the presence of up to $2d$ malicious nodes, or (ii) identification of up to d malicious nodes.

Proof: The proof of (i) is by contradiction. If $2d$ (or less) malicious nodes, say nodes i_1, i_2, ..., i_{2d} that generate nonzero errors $\varepsilon_{i_1}[K]$, $\varepsilon_{i_2}[K]$, ..., $\varepsilon_{i_{2d}}[K]$, go undetected, the syndrome $p[K]$ satisfies $p[K] = \sum_{l=1}^{2d} C(:, i_l)\varepsilon_{i_l}[K] = 0$, where $C(:, i_l)$ denotes the i_lth column of C. However, this contradicts the assumption that any $2d$ columns of C are linearly independent.

The proof of (ii) is also by contadiction. Suppose that two different sets of d columns produce the same syndrome, i.e., $\sum_{l=1}^{d} C(:, i_l)\varepsilon_{i_l}[K] = \sum_{l=1}^{d} C(:, j_l)\varepsilon_{j_l}[K]$ where i_l and $\varepsilon_{i_l}[K]$ (for $l = 1, 2, ..., d$) correspond to one set of malicious nodes and j_l and $\varepsilon_{j_l}[K]$ (for $l = 1, 2, ..., d$) correspond to the other set of malicious nodes. Then,

$$\sum_{l=1}^{d} C(:, i_l)\varepsilon_{i_l}[K] - \sum_{l=1}^{d} C(:, j_l)\varepsilon_{j_l}[K] = 0$$

is a linear combination that evaluates to zero and involves at most $2d$ columns (perhaps less if some of them are duplicates and their coefficients cancel out). Thus, we have reached a contradiction. ■

Notice that if $-C$ is chosen to be a Vandermonde matrix

$$V(y_1, y_2, ..., y_n) = \begin{bmatrix} y_1 & y_2 & \ldots & y_n \\ y_1^2 & y_2^2 & \ldots & y_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ y_1^{2d-1} & y_2^{2d-1} & \ldots & y_n^{2d-1} \\ y_1^{2d} & y_2^{2d} & \ldots & y_n^{2d} \end{bmatrix} \quad (10)$$

with $y_i \neq 0$ for $1 \leq i \leq n$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$, then any set of $2d$ columns of it will be linearly independent. In fact, in such case one can use the techniques in [6] to identify the malicious nodes. For example, apart from exhaustively searching over all combinations of $2d$ columns of $-C$ (to find d or less of them that can be linearly combined to obtain the parity $p[K]$), one can also use a variant of the Petterson-Gorenstein-Zierler decoding algorithm (used in Galois fields for decoding Reed-Solomon codes [14]) to efficiently determine the d' errors based on the syndrome $p[K]$ [6].

C. Two-Hop Information Received every K Iterations

In the redundant implementation in (4) with matrices $\mathcal{A}$ and $\mathcal{B}$ as in (6), each node v_i in the set $\{v_1, v_2, ..., v_n\}$ is in charge of updating its own state variable $x_i[k]$ and clearly has access to the variables in $u[k]$ that it needs (node v_i is updating $x_i[k]$ as in (2) using information from its immediate neighbors). However, the last $2d$ variables of $r[k]$ are actually internal to node v_1 which is implementing the checking scheme. Node v_1 clearly has access to $r[k]$ but requires two-hop transmissions to maintain access to $u[k]$. To relax this requirement, we have node v_1 treat $u[k]$ as zero[1] at iterations $0, 1, ..., K-1$, so that the update (using the choices in (6) and $D = I_{2d}$) becomes

$$r'[k+1] = \left[\begin{array}{c|c} A & 0 \\ \hline CA - C & I_{2d} \end{array}\right] r'[k] + \left[\begin{array}{c} B \\ \hline 0 \end{array}\right] u[k] \ , \quad (11)$$

with an (additional) error of the form

$$\delta[k] = -\left[\begin{array}{c} 0 \\ \hline CB \end{array}\right] u[k]$$

introduced at each iteration k.

Clearly, with the above updates, the error vector at the end of the time window becomes

$$\varepsilon'[K] = \underbrace{\begin{bmatrix} 0 \\ \varepsilon_2[K] \\ \vdots \\ \varepsilon_n[K] \\ 0_{2d} \end{bmatrix}}_{=\varepsilon[K]} - \underbrace{\begin{bmatrix} 0_n \\ CB\sum_{t=0}^{K-1} u[t] \end{bmatrix}}_{=\sum_{t=0}^{K-1}\delta[t]} , \quad (12)$$

where 0_{2d} (0_n) is the $2d$-dimensional (n-dimensional) vector of all zeros.

Furthermore, if the two-hop neighbors of node v_1 provide the values $\sum_{t=0}^{K-1} u_j[t]$ for $j = 1, 2, ..., m$ (i.e., each two-hop neighbor v_{n+j} provides the cumulative sum of its x_{n+j} values over the time window), then node v_1 can easily compute the adjusted syndrome $p'[K]$ as

$$p'[K] = p[K] + Q\left(\sum_{t=0}^{K-1}\delta[t]\right) = Qr'[K] + CB\sum_{t=0}^{K-1}u[t] \ .$$

[1]Other choices of $u[k]$ are also possible, as long as proper adjustments are made once the cummulative sum of $u[k]$ is received at iteration K.

This adjusted syndrome satisfies

$$p'[K] = [-C \; I_{2d}]\, \varepsilon[K] = -C \underbrace{\begin{bmatrix} 0 \\ \varepsilon_2[K] \\ \vdots \\ \varepsilon_n[K] \end{bmatrix}}_{=\widetilde{\varepsilon}[K]}.$$

Therefore, if any $2d$ columns of C are linearly independent and the error vector $\widetilde{\varepsilon}[K]$ has at most d', $d' \leq d$, nonzero entries, then these entries can be detected and identified based on the syndrome $p'[k]$. To do this, one can use any of the techniques described at the end of the previous section (e.g., define $-C$ to be a Vandermonde matrix with distinct nonzero parameters, etc.). The proof of the following lemma is omitted as it follows similar steps as the proof of Lemma 1.

Lemma 2: Suppose we build the monitor in (11) with matrix C having the property that any $2d$ columns of it are linearly independent. Furthermore, assume that each malicious node v_i is associated with a nonzero $\varepsilon_i[K]$ and that two-hop neighbors of node v_1 correctly report their cumulative sums at time instant K in (12). Then, we can guarantee (i) detection of the presence of up to $2d$ malicious nodes within the neighborhood of node v_1, or (ii) identification of up to d malicious nodes within the neighborhood of node v_1.

It is important to point out that the above analysis assumes that the cumulative sums $\sum_{t=0}^{K-1} u_j[t]$, $j = 1, 2, ..., m$, are reported *correctly* to node v_1 by its two-hop neighbors v_{n+j}, respectively, at iteration K. This is not a strong assumption because even though node v_1 is not in position to determine if node v_{n+j} reports an erroneous cumulative sum $\sum_{t=0}^{K-1} u_j[t]$, all of the (immediate) neighbors of node v_{n+j} can easily flag this. In our future work, we plan to explore approaches that can also handle erroneous reporting by two-hop neighbors (by allowing additional errors, beyond errors in $\widetilde{\varepsilon}[K]$, to be introduced in the adjusted syndrome $p'[k]$).

D. Discussion

The proposed error detection and identification scheme groups together the total error $\varepsilon_i[K]$ because the objective is to identify whether node v_i is malicious. This is not restrictive in the sense that if node v_i is identified as malicious, i.e., node v_i is associated with a nonzero $\varepsilon_i[K]$, then the effect of node v_i on the computation can be corrected without having to determine the individual errors that node v_i injected in the computation—all that is needed is to subtract $\varepsilon_i[K]$ from the state variable of node v_i and continue the average consensus iteration (see, e.g., [15]).

It is worth noting that node v_i will not be identified as malicious if it introduces errors in the interval $[0, K-1]$ but manages to have $\varepsilon_i[K] = 0$ at the end of the time window; however, this also implies that the errors node v_i introduces are not going to affect the outcome of the average consensus scheme. This approach should be contrasted with the approach in [6], which aims at identifying individual errors $e_i[t]$ (for some t's in $[0, K-1]$). Also note that there are ways to deal with malicious nodes that, in order to remain undetected, inject errors such that the total error is zero at multiples of the checking period K. One easy way to do this is to randomize the length of the checking window (K) by having each checker node randomly determine when to request accumulated values from its two-hop neighbors.

Before closing these discussions, we should point out that in practice both the average consensus and checking processes will have to deal with finite precision issues. Thus, one would have to carefully decide whether small nonzero values in the parity vector $p[K]$ are due to the presence of a malicious node or simply due to finite precision limitations. Some discussions and references on such issues can be found in [6]. For example, the parameters y_1, y_2, ..., y_n of the Vandermonde matrix in (10) could be chosen to be close to unity (or, if one is willing to deal with complex values, equally spaced on the unit circle).

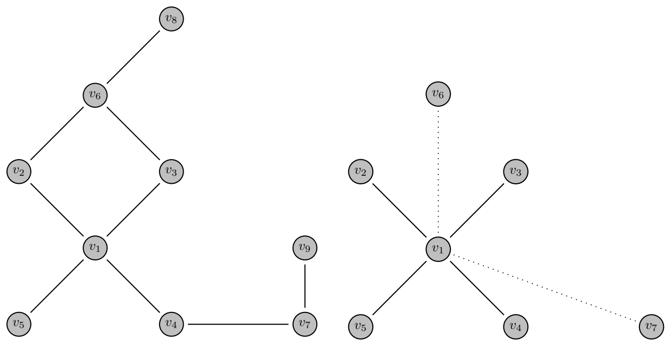

Fig. 1. Left: Undirected graph $\mathcal{G}_u$ considered in the example. Right: Connections for the monitor implemented at node v_1 (dotted lines indicate periodic transmissions from two-hop neighbors).

IV. Illustrative Example

Consider the undirected graph $\mathcal{G}_u = (\mathcal{V}, \mathcal{E})$ of order $N = 9$, with $\mathcal{V} = \{v_1, v_2, ..., v_9\}$ and edges as shown on the left of Fig. 1. The initial values of the nodes are taken to be $x[0] = [x_1[0],\ x_2[0],\ ...,\ x_9[0]]^T = [1,\ 2,\ ...,\ 9]^T$ so that the average is $\overline{V} = 5$. We assume that nodes know an upper bound $N' = 10$ on the number of nodes and use weights 0.1 on all edges (and appropriate self-weights).

We now describe the non-concurrent scheme from the point of view of node v_1; however, please keep in mind that *each* node in the network is doing something analogous. The neighbors of node v_1 are $\mathcal{N}_1 = \{v_2, v_3, v_4, v_5\}$, whereas the set $\mathcal{N}_1^{(2)} \setminus (\mathcal{N}_1 \cup \{v_1\}) = \{v_6, v_7\}$ (the subnetwork associated with node v_1 as a checker node is shown on the right of Fig. 1). Therefore, in (3), we have $n = 5$, $m = 2$, $q[k] = [x_1[k],\ x_2[k], ...,\ x_5[k]]^T$; and $u[k] = [x_6[k],\ x_7[k]]^T$, with matrices A and B given by

$$A = \begin{bmatrix} 0.6 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0 & 0 & 0 \\ 0.1 & 0 & 0.8 & 0 & 0 \\ 0.1 & 0 & 0 & 0.8 & 0 \\ 0.1 & 0 & 0 & 0 & 0.9 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0.1 & 0 \\ 0.1 & 0 \\ 0 & 0.1 \\ 0 & 0 \end{bmatrix}.$$

We choose $d = 2$ so that node v_1 can detect and identify up to two malicious nodes, and we set

$$C = -V(1, -1, 2, -2, 3) = -\begin{bmatrix} 1 & -1 & 2 & -2 & 3 \\ 1 & 1 & 4 & 4 & 9 \\ 1 & -1 & 8 & -8 & 27 \\ 1 & 1 & 16 & 16 & 81 \end{bmatrix}$$

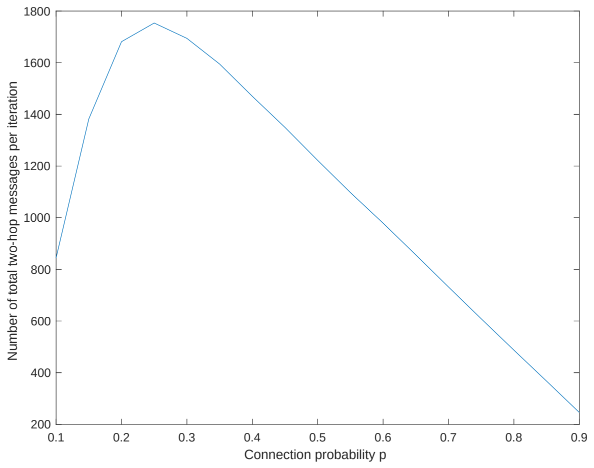

Fig. 2. Average number of two-hop messages saved at each iteration.

(i.e., we choose $y_1 = 1$, $y_2 = -1$, $y_3 = 2$, $y_4 = -2$, and $y_5 = 3$) so that matrices $\mathcal{A}$ and $\mathcal{B}$ can be obtained as in (11). The redundant state is initialized to $r[0] = [1, 2, 3, 4, 5, -12, -76, -126, -520]^T$.

To illustrate how the nonconcurrent scheme works, suppose we take $K = 10$ and run the iteration in (11) for $k = 0, 1, ..., 9$. Furthermore, assume that the values of $u_1[0]$, $u_1[1]$, ..., $u_1[9]$ are such that $\sum_{t=0}^{9} u_1[t] = 45.19$, and the values of $u_2[0]$, $u_2[1]$, ..., $u_2[9]$ are such that $\sum_{t=0}^{9} u_2[t] = 59.98$ (these values could be arbitrary).

Assume that node v_3 is malicious and adds additive errors $e_3[0]$, $e_3[1]$, ..., $e_3[9]$ to its value so that $\sum_{t=0}^{9} e_3[t] = 11$. We update $r'[K]$ as in (11) and at $K = 10$, we have $r'[10] = [5.11,\ 2.99,\ 8.71,\ 5.14,\ 4.24,\ -7.46,\ -11.06,\ -73.52,\ -224.32]^T$. We calculate $p[10] = Qr'[10]$ and obtain $p'[10] = \left[\ 22,\ \ 44,\ \ 88,\ \ 176\ \right]^T$; as expected $p'[10] = 11 \times V(:,3)$ where $V(:,3)$ is the third column of matrix V. This implies that a total error of 11 has been added to the variable x_3 by node v_3.

We also provide a simulation study to illustrate the benefits of the proposed approach in terms of the reduction on the number of two-hop messages that are required. More specifically, we consider random undirected graphs of order $N = 50$, where an edge is present between a pair of nodes with probability p (independently among different pairs of nodes). In Fig. 2, we plot, the sum (over all nodes v_j) of the number of two-hop neighbors that are not direct neighbors of node v_j. This number is directly related to the number of messages to two-hop neighbors that are saved at each iteration by using non-concurrent checking. We generate each point for a particular value of p by averaging over 100 random graphs. As we can see, for $p = 0.3$, each node saves on average close to 36 messages per iteration. As expected, the savings diminish as the graphs get denser.

V. Conclusions

In this paper, we have considered the problem of distributed average consensus in the presence of malicious nodes that may try to influence the outcome of the computation via incorrect updates. The proposed algorithm allows each node v_j to check its neighboring nodes based on information that it receives from them directly, as well as information that it receives periodically by their neighbors (i.e., by the neighbors of its neighbors). The main advantage of the proposed scheme is that it ensures that two-hop information is only needed periodically, which significantly relaxes the communication overhead.

In the future, we plan to explore ways to detect and identify nodes that enter nonzero errors at certain time steps but try to avoid detection by ensuring that their cumulative error is zero. We also plan to explore ways to handle incorrect information provided by two-hop neighboring nodes. Another interesting future direction is towards switching (undirected) topologies, which can take advantage of non-concurrent error correction schemes in switched linear systems [16]. Utilizing two-hop information to simultaneously perform checking and speed-up convergence to the average is also an interesting future direction.

References

[1] J. Tsitsiklis, "Problems in decentralized decision making and computation," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[2] C. N. Hadjicostis, A. D. Domínguez-García, and T. Charalambous, "Distributed averaging and balancing in network systems, with applications to coordination and control," *Foundations and Trends® in Systems and Control*, vol. 5, no. 3–4, 2018.

[3] N. A. Lynch, *Distributed Algorithms*. San Mateo: CA: Morgan Kaufmann Publishers, 1996.

[4] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[5] J. Cortés, "Distributed algorithms for reaching consensus on general functions," *Automatica*, vol. 44, no. 3, pp. 726–737, 2008.

[6] C. N. Hadjicostis, "Nonconcurrent error detection and correction in fault-tolerant discrete-time LTI dynamic systems," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 1, pp. 45–55, 2003.

[7] L. Yuan and H. Ishii, "Resilient consensus with distributed fault detection," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 285–290, 2019.

[8] ——, "Resilient consensus with multi-hop communication," in *Proc. of the 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 2696–2701.

[9] ——, "Secure consensus with distributed detection via two-hop communication," *Automatica*, vol. 131, p. 109775, 2021.

[10] M. Yemini, A. Nedic, A. J. Goldsmith, and S. Gil, "Characterizing trust and resilience in distributed consensus for cyberphysical systems," *IEEE Trans. on Robotics*, vol. 38, no. 1, pp. 71–91, 2021.

[11] C. N. Hadjicostis and A. D. Domínguez-García, "Trustworthy distributed average consensus," in *Proc. of the 61st IEEE Conference on Decision and Control (CDC)*, 2022, pp. 7403–7408.

[12] H. J. LeBlanc, H. Zhang, S. Sundaram, and X. Koutsoukos, "Consensus of multi-agent networks in the presence of adversaries using only local information," in *Proc. of the 1st International Conference on High Confidence Networked Systems*, 2012, pp. 1–10.

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

[14] R. E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983, vol. 126.

[15] N. E. Manitara and C. N. Hadjicostis, "Privacy-preserving asymptotic average consensus," in *Proc. of European Control Conference (ECC)*, 2013, pp. 760–765.

[16] S. Sundaram and C. N. Hadjicostis, "Error detection and correction in switched linear controllers via periodic and non-concurrent checks," *Automatica*, vol. 42, no. 3, pp. 383–391, 2006.