A Fault-Tolerant Distributed Termination Method for Distributed Optimization Algorithms

Mohannad Alkhraijah and Daniel K. Molzahn

***Abstract*—This paper proposes a fully distributed termination method for distributed optimization algorithms solved by multiple agents. The proposed method guarantees terminating a distributed optimization algorithm after satisfying the global termination criterion using information from local computations and neighboring agents. The proposed method requires additional iterations after satisfying the global terminating criterion to communicate the termination status. The number of additional iterations is bounded by the diameter of the communication network. This paper also proposes a fault-tolerant extension of this termination method that prevents early termination due to faulty agents or communication errors. We provide a proof of the method's correctness and demonstrate the proposed method by solving the optimal power flow problem for electric power grids using the alternating direction method of multipliers.**

***Index Terms*—Distributed Optimization, Fault Tolerance, Termination Criteria.**

I. Introduction

Distributed optimization algorithms allow multiple computing agents to collaboratively solve large-scale optimization problems by solving smaller subproblems. Distributed optimization algorithms are typically iterative, where agents solve the subproblems in parallel, share their solutions at each iteration, and terminate after finding a solution that satisfies predefined termination criteria. The potential advantages of solving large-scale optimization problems using distributed algorithms include improving scalability by allowing parallel computation, reliability by eliminating a single point of failure, and privacy by reducing the amount of shared data [1].

The advantages of using distributed optimization algorithms motivate their application in electric power systems [2], wireless sensor networks [3], robotics [4], machine learning [5], etc. Realizing all potential benefits of distributed optimization in many of these applications requires a fully distributed termination method that eliminates the need for a central coordinator to check the termination statuses of all agents. However, to the best of our knowledge, the literature lacks a fully distributed method for terminating general distributed optimization algorithms.

A fully distributed optimization algorithm implementation requires the use of a distributed termination method. By *distributed termination*, we mean a method that allows agents to agree that a predefined global termination criterion is satisfied and the algorithm can thus terminate despite only having their local information and data communicated with immediate neighbors. Finding an appropriate termination method is a challenging problem in distributed computing in general because no agent has direct access to the global system's convergence status [6]. In addition, it is desirable that termination methods for distributed optimization algorithms are symmetrical with common rules across all agents, require minimal prior knowledge of the system, and have no restrictions on the communication network topology.

A. Related Work

Terminating computations has been extensively studied in the context of distributed computing. The first methods for terminating distributed computations run in a combination of series and parallel sequences were independently proposed in [7] and [8]. In the distributed termination problem of distributed computing, there are multiple processors, each with a predecessor and successors. A processor is active when it receives a message from a predecessor. Once the processor completes its computations, the processor becomes an idle processor until a new message is received. A termination method stops the computation when all the processors are idle and no messages are in transit [6].

Several distributed termination methods have been proposed in the literature to terminate distributed computing processors [9]. Reference [8] proposes a method that uses a rooted tree to track the global termination status. Another widely used approach proposed in [10] uses credit distribution, where each processor receives a credit from its predecessors when activated, and returns the credit when the computations are completed. Another family of distributed termination methods uses a wave-like message passing of the termination status [7], [11]. Reference [11] proposes using a wave-based method that passes clock-like information and requires defining a Hamiltonian cycle that traverses all processors. A similar method proposed in [12] uses a predefined ring network to pass the termination status across the processors.

A method proposed in [13] combines ideas from [8] and [11] by using a forest, i.e., multiple rooted trees, in addition to a wave circulating between the trees' roots. A method proposed in [14] uses a combination of credit distribution and a rooted tree, where the root processor maintains a ledger and counts the credits of the successors and predecessors in each level of the tree. Reference [15] proposes a "vector method" where a vector containing information about the termination status of each processor is communicated through a ring network until the termination criteria are satisfied.

An important property for distributed termination methods is *fault tolerance*, i.e., the ability of an algorithm to operate reliably even with the existence of a faulty processor [9]. A fault-tolerant method proposed in [16] uses multiple message passing at the same time to prevent faulty termination. The

authors of [17] propose using a variation of the rooted tree method by having multiple roots of the rooted tree and a mechanism for reorganizing the tree's nodes. Most recently, [18] proposes a modification of the ring method in [12] that regularly checks the processors' statuses and reorganizes the ring's network upon detecting a faulty processor.

Although these distributed termination methods are widely used in distributed computing, they are not suitable for distributed optimization algorithms. The main difference between distributed optimization and distributed computing is that the agents in distributed optimization algorithms run simultaneously, i.e., there are no successors and predecessors, and no agent will be idle until the algorithm terminates. Further, fully distributed optimization algorithms may not have a central agent that initiates and terminates the algorithm similar to [8], [10], [14], nor a communication network topology restriction similar to [11]–[13], [15], [18]–[20].

The only approach we are aware of that roughly discusses a similar problem is proposed in [21] to solve distributed model predictive control problems. The method in [21] uses a gossip-based method [22] to calculate the global termination criteria. Although the method is more suitable for distributed control than optimization, the same concept could be extended to distributed optimization algorithms. However, extending the method requires a problem-specific formulation to design the local termination criteria. Moreover, the method in [21] solves the gossip-based algorithm after each iteration, thus adding a significant computational and communications burden, and susceptible to faults that may cause the algorithm to terminate before satisfying the termination criterion.

B. Contributions

This paper proposes a fully distributed termination method for a general distributed optimization algorithm. The proposed method is fully distributed and guaranteed to terminate the algorithm after the global termination criterion is satisfied. The method uses simple rules that are symmetrical across all agents, requires minimal prior knowledge of the system, and does not impose any restrictions on the communication network topology. Furthermore, we propose an extension to the method that is fault-tolerant to early termination. The paper presents a proof of the method's correctness as well as numerical verification.

To summarize, this paper:

- Motivates the importance of a fully distributed termination method for the practical implementation of distributed optimization algorithms,
- Proposes a termination method for fully distributed optimization algorithms,
- Provides a fault-tolerant extension that prevents early termination caused by a fault,
- Proves the correctness of the proposed method and the fault-tolerant extension, and
- Demonstrates the effectiveness of the proposed method using practical examples of optimal power flow problems for distributed optimization of electric power grids.

C. Organization

The rest of the paper is organized as follows: Section II presents the mathematical notations and defines the distributed optimization termination problem. Section III presents the proposed method and the correctness proof. Section IV describes a variant of the proposed method that prevents faulty termination. Section V shows test cases and numerical verification. Section VI provides remarks and extensions, and Section VII presents conclusions.

II. Distributed Termination Problem

This section introduces notation and states the distributed termination problem with the help of several definitions.

A. Notation

We represent the communication network with a graph $G(\mathcal{A}, \mathcal{E})$, where $\mathcal{A}$ is the set of agents and $\mathcal{E}$ is the set of communication links. We use $\mathcal{N}(i) \subset \mathcal{A}$ to denote $\{j \mid (i,j) \in \mathcal{E}\}$, the set of neighboring agents that are connected to agent i by a communication link (excluding agent i itself). We define a path between agent i and j as $P(i,j) := \{k_0, k_1, k_2, \ldots, k_N\}$, a sequence of distinct agents such that $k_0 = i, k_N = j$, $(k_n, k_{n+1}) \in \mathcal{E}$, and the path length N, where the path length is the number of communication links in the path. We define the distance $d(i,j)$ as the length of the shortest path between agents i and j and the diameter of a network as $D := \max_{i,j \in \mathcal{A}} \{d(i,j)\}$, the maximum distance among all the pairs of agents. We denote the iteration index with $t \in \mathcal{T}$, where $\mathcal{T} = \{0, 1, 2, 3, \ldots, \overline{T}_G\}$ is the set of iterations before the algorithm terminates, and $\overline{T}_G$ is the maximum number of iterations. We use $|\cdot|$ to denote the cardinality of a set, and $\bigcup$ to denote the union of two sets. We use $\mathcal{A} \setminus \mathcal{A}'$ to indicate the set of agents in $\mathcal{A}$ and not in $\mathcal{A}'$. For any general variable X, the superscript t in X^t denotes variables that are known at iteration t, the subscript i in X_i denotes variables belonging to agent i, and subscript G in X_G denotes global variables that no agent can directly access. If X is a vector, then $X[i]$ is a scalar that denotes the i-th entry of the vector X.

B. Problem Statement and Definitions

Consider multiple agents solving an optimization problem using an iterative distributed optimization algorithm. The distributed agents are connected via a communication network represented by an undirected graph $G(\mathcal{A}, \mathcal{E})$, where $\mathcal{A}$ and $\mathcal{E}$ are the sets of agents and communication links. At each iteration, the agents perform local computations and share information about their local status via the communication network. Agent i communicates with agent j if and only if $(i,j) \in \mathcal{E}$. At the end of each iteration, the agents decide whether to terminate the algorithm or proceed to the next iteration based on local rules. The distributed optimization algorithm should only terminate after the agents satisfy a global termination criterion. We have the following definitions:

Definition 1 (D1)**.** *Local termination criterion*: A condition defining when an agent reaches consensus on the computation

results with the neighboring agents. We denote the status of the local termination criterion of agent i at iteration t by $B_i^t \in \{0,1\}$, where $B_i^t = 1$ when agent i satisfies the local termination criterion at iteration t, and $B_i^t = 0$ otherwise.

The predefined condition of the local termination criterion involves reaching a consensus with neighboring agents on some shared quantities within a predefined tolerance. These quantities are typically shared variables' values or the amount by which coupling constraints between neighboring agents are violated. Shared quantities may also include dual residuals that measure the optimality of the solutions. Agents measure consensus using a norm of the mismatch or violations with the neighboring agents. See, e.g., [23, Section 3.3] for an example of a local termination criterion. Each agent evaluates their local termination criterion using available information from local computations and data communicated with immediate neighbors.

Definition 2 (D2)**.** *Global termination criterion*: A condition defining when all of the agents reach consensus on the distributed optimization results. The global termination criterion is satisfied when all agents satisfy their local termination criteria. We denote the status of the global termination criterion at iteration t by $B_G^t \in \{0,1\}$, where $B_G^t = 1$ when all agents satisfy their local termination criteria at iteration t (i.e., $B_i^t = 1$, for all $i \in \mathcal{A}$), and $B_G^t = 0$ otherwise.

Satisfying the global termination criterion indicates that the distributed optimization algorithm has reached the desired results, and agents should terminate the computations. Since the global termination criterion depends on information from every agent, its status is not directly available to any of the agents. Thus, agents should communicate the global termination criterion and agree on terminating the algorithms.

Definition 3 (D3)**.** *Appropriate algorithm termination*: The agents simultaneously terminate their computations after the global termination criterion is satisfied.

In other words, we say that an algorithm terminates *appropriately* if the following two conditions are satisfied upon termination: 1) the global termination criterion is satisfied (i.e., all agents' local termination criteria are satisfied) and 2) all agents terminate their computations simultaneously. More formally, if the global termination criterion is satisfied at iteration T_G, then an appropriate algorithm termination implies that $T_G \leq \overline{T}_i = \overline{T}_G$, for all $i \in \mathcal{A}$, where $\overline{T}_i$ is the final iteration of agent i and $\overline{T}_G$ is the last iteration of any agent. Thus, the algorithm may still terminate appropriately if the agents continue their computations for additional iterations after the global termination criterion is satisfied. While the additional $\overline{T}_G - T_G$ iterations are not necessary for the convergence of the distributed algorithm, they may be needed to ensure an appropriate algorithm termination.

III. Distributed Termination Method

This section presents our proposed distributed termination method. This method is based on a set of rules that agents apply at each iteration to either certify that the global termination criterion has been met and they can thus terminate or that they should proceed to the next iteration. The agents apply these rules after the communication step of the distributed algorithm, prior to starting a new iteration. These rules depend strictly on information available to each agent via local computations and communication with neighboring agents. The proposed method combines ideas from [11], in the sense of passing the termination status using a wave-like message with a clock-like stamp, and the vector method from [15] since the agents' termination statuses are stored in a vector.

Our method relies on the following three assumptions:

Assumption 1 (A1)**.** The communication network graph is connected, i.e., for each pair of agents i and $j \in \mathcal{A}$, there is a path $P(i,j)$.

This assumption implies that information communicated by an agent can traverse all other agents.

Assumption 2 (A2)**.** All agents know the total number of agents $|\mathcal{A}|$ and the diameter of the communication network D. No other prior knowledge is required.

This assumption implies that agents have a minimal prior knowledge of the communication network topology and the total number of other agents. The only additional knowledge each agent has comes from their local computations and the information communicated by neighboring agents.

Assumption 3 (A3)**.** The local termination criterion is defined by a monotonic process, i.e., $B_i^t \leq B_i^{t'}$ if $t \leq t'$.

In other words, after being satisfied, an agent's local termination criterion remains satisfied for all subsequent iterations.

A. Method Procedure

We next present our proposed distributed termination method. Each agent i stores a local vector $V_i \in \{0,1\}^{|\mathcal{A}|}$, called a *termination vector*, and a local scalar $T_i \in \mathcal{T}$, called a *termination iteration scalar*. We call the combination of both V_i and T_i the *termination status* for agent i. The termination vector V_i stores agent i's information about the satisfaction of every agents' local termination criterion. The termination iteration scalar T_i stores agent i's information regarding the iteration number when the last agent satisfies its local termination criterion. Let t be the current iteration and B_i^t be the status of the local termination criterion of agent i at iteration t as defined in Definition D1. At $t = 0$, agent i uses the following rule to initialize its termination status:

Rule 0 (R0)**.** Set $T_i^0 = 0$ and $V_i^0[k] = 0$ for all $k \in \mathcal{A}$.

At iteration $t \geq 1$, agent i updates its local termination criterion's status B_i^t and receives the termination statuses V_j^{t-1} and T_j^{t-1} from neighboring agents $j \in \mathcal{N}(i)$. Then, agent i applies the following rules:

Rule 1 (R1)**.** Set $T_i^t = \max_{j \in \mathcal{N}(i)} \left\{T_j^{t-1}\right\}$ and $V_i^t[k] = \max_{j \in \mathcal{N}(i)} \left\{V_j^{t-1}[k]\right\}$, for all $k \in \mathcal{A} \setminus \{i\}$.

Rule 2 (R2)**.** If $V_i^{t-1}[i] = 0$ and $B_i^t = 1$, then set $T_i^t = t$ and $V_i^t[i] = 1$.

Rule 3 (R3)**.** If $\sum_{k\in\mathcal{A}} V_i^t[k] = |\mathcal{A}|$ and $t \geq T_i^t + D$, terminate.

Recall that D in Rule R3 is the diameter of the communication network, which we assume each agent knows due to Assumption A2. Rule R1 uses communications from neighboring agents to update an agent's local termination status. Rule R2 uses an agent's local information to update its local termination status when the local termination criterion is first satisfied. Rule R3 dictates when the agents terminate their computations. The four rules R0–R3 ensure that the distributed optimization algorithm will terminate appropriately after $T_G + D$ iterations. In other words, the agents will simultaneously terminate computations at D iterations (diameter of the communication network) after satisfying the global termination criterion (which occurs at iteration T_G). Using this method, the agents only need to store a binary vector of size $|\mathcal{A}|$ ($V_i \in \{0,1\}^{|\mathcal{A}|}$) and a single non-negative integer ($T_i \in \mathbb{Z}_{\geq 0}$).

B. Method Correctness

The algorithm is *correct* if the four rules R0–R3 ensure that the algorithm will appropriately terminate; that is, agents following R0–R3 terminate simultaneously after satisfying the global termination criterion. To prove the correctness of the algorithm, we use the following three propositions.

Proposition 1 (P1)**.** For any agents i and $j \in \mathcal{A}$, the j-th entry of the termination vector of agent i is one (i.e., $V_i^t[j] = 1$) if and only if agent j satisfies its local termination criterion (i.e., $B_j^t = 1$).

Proof. For the forward implication, since the value of the termination vector is initialized as $V_i^0[j] = 0$, for all $i, j \in \mathcal{A}$ due to Rule R0 and $V_i^t[j] = 1$, then the value of $V_i^t[j]$ changes at or before iteration t. Without loss of generality, assume $V_{i_0}^{t_i-1}[j] = 0$ and $V_{i_0}^{t_i}[j] = 1$ at iteration $t_i \leq t$ for some agent $i_0 \in \mathcal{A}$. This change occurs due to either Rule R1 or Rule R2. If this change occurs due to Rule R2, then $i_0 = j$ since Rule R2 only changes the same agent's status, and we can thus directly conclude from Rule R2 that agent j satisfies its local termination criterion at iteration t_i (i.e., $B_j^{t_i} = 1$). Since $t_i \leq t$, then $B_j^t = 1$ due to Assumption A3, which completes the forward implication for this case. Alternatively, if the change occurs due to Rule R1, then $i_0 \neq j$ since Rule R1 only changes the status associated with another agent. This implies $V_{i_1}^{t_i-2}[j] = 0$ and $V_{i_1}^{t_i-1}[j] = 1$, for some agent $i_1 \in \mathcal{N}(i_0)$. Using a similar argument to the one we used for agent i_0, we conclude either agent $i_1 = j$, which implies $B_j^t = 1$, or there is $V_{i_2}^{t_i-3}[j] = 0, V_{i_2}^{t_i-2}[j] = 1$, for some agent $i_2 \in \mathcal{N}(i_1)$ and $i_2 \notin \mathcal{N}(i_0) \bigcup \{i_0\}$. We recursively use the same argument to obtain a sequence $\mathcal{I} = \{i_0, i_1, i_2, \ldots\} \subseteq \mathcal{A}$. The sequence $\mathcal{I}$ is finite since $\mathcal{A}$ is a finite set and each agent can appear no more than once. Consider the last agent in $\mathcal{I}$, denoted as agent i', with $V_{i'}^{t_{i'}}[j] = 1$ and $t_{i'} = t_i - (|\mathcal{I}| - 1) \leq t$. Agent i' must have used Rule R2 to update $V_{i'}^{t_{i'}}[j]$, since it is the last agent in $\mathcal{I}$; otherwise, if i' used Rule R1, then i' would not be the last agent in $\mathcal{I}$. Since i' used Rule R2, then $i' = j$ and $B_j^{t_{i'}} = 1$. Thus, $B_j^t = 1$ due to Assumption A3 and $t_{i'} \leq t$, which completes the proof of the forward implication. The backward implication is a direct consequence of Rule R2. If agent j satisfies the local termination criterion $B_j^t = 1$, then $V_j^t[j] = 1$ due to Rule R2. □

Proposition 2 (P2)**.** If, for any agent $i \in \mathcal{A}$, all terms in the termination vector are equal to one (i.e., $\sum_{k\in\mathcal{A}} V_i^t[k] = |\mathcal{A}|$), then the distributed optimization algorithm satisfies the global termination criterion (i.e., $B_G^t = 1$).

Proof. Using Proposition P1, if $\sum_{k\in\mathcal{A}} V_i^t[k] = |\mathcal{A}|$, then all agents satisfy their local termination criterion. Thus, the agents satisfy the global termination criterion by Definition D2. □

Note that the reverse is not necessarily true since we might have a case where all the agents satisfy their local termination criteria, but some agents have not received the updated local termination status yet.

Proposition 3 (P3)**.** If the agents satisfy the global termination criterion at iteration T_G, then the algorithm terminates appropriately in $T_G + D$ iterations.

Proof. Without loss of generality, consider the case when all the agents satisfy their local termination criteria at iteration t' except for agent i, i.e., $B_j^{t'} = 1$, for all $j \in \mathcal{A} \setminus \{i\}$. If agent i satisfies the local termination criterion at iteration T_G, then $T_i^t = T_G$, for all $t \geq T_G$ due to Rule R2 and Assumption A3. By communicating T_G and using Rule R1 to update the iteration termination scalar in the next iterations by all agents, we get $T_j^t = T_G$, for all $j \in \mathcal{A}, t_j \geq T_G + D$, since the number of iterations needed to communicate a status update between agents i and j is equal to the distance between the agents $d(i,j)$ and the distance between any two agents is less than or equal to the diameter of the communication network, i.e., $d(i,j) \leq D$. The analogous argument applies to updates for $V_j^t[k]$, for all $j, k \in \mathcal{A}$. This implies $T_j^t = T_G$ and $V_j^t[k] = 1$, for all $j, k \in \mathcal{A}, t \geq T_G + D$. Thus, all agents receive the updated global termination criterion status using Proposition P2 and terminate appropriately at iteration $T_G + D$ using Rule R3. □

The last proposition shows that the proposed method terminates the distributed optimization appropriately and thus the method is correct.

IV. Fault-Tolerant Distributed Termination Method

The termination method described in Section III assumes all the agents are reliable and share accurate termination statuses. However, agents may fail to properly update termination statuses due to communication errors or other faults. Not following the update rules may either cause early termination or failure to terminate. This section extends the method in Section III to prevents early termination. Formally, we define:

Definition 4 (D4)**.** *Faulty termination* (also called *early termination)*: One or more agents terminate their computations before the global termination criterion is satisfied.

Faulty termination could occur if there were faulty agents not following Rule R1 correctly by assigning a false positive or false negative termination status to one or more entries in

their termination vector. Thus, Proposition P1 would not be true. We use the following definition:

Definition 5 (D5)**.** *Fault injection*: Any update to the termination status that does not follow the termination method's rules.

For notational convenience, we call an agent that injects a faulty status a *faulty* agent and an agent whose termination status is incorrectly reported a *faulted* agent. That is, if agent i falsely reports that agent j has reached its termination criterion in violation of Rule R1, we say that agent i is faulty and agent j is faulted.

Before stating additional rules, we identify two sufficient conditions that indicate the existence of a faulty termination status. We then use these two conditions to correct any faulty termination status in the fault-tolerant method.

Proposition 4 (P4)**.** If, for some agents i and $k \in \mathcal{A}$ and iteration t, the k-th entry of the termination vector of agent i is $V_i^{t-1}[k] = 1$ and the termination vector for some neighboring agent $j \in \mathcal{N}(i)$ is $V_j^t[k] = 0$, then there is a faulty termination status.

Proof. Assume Proposition P4 is not true, that is, there is no faulty termination status, $V_i^{t-1}[k] = 1$, and $V_j^t[k] = 0$, $j \in \mathcal{N}(i)$ for some i and $k \in \mathcal{A}$. If $V_i^{t-1}[k] = 1$, then $V_j^t[k] = 1$, for all $j \in \mathcal{N}(i)$ due to Rule R1, which contradicts our assumption that there is $j \in \mathcal{N}(i)$ with $V_j^t[k] = 0$. Thus, there is a faulty termination status. □

Proposition 5 (P5)**.** If, for some agents i and $k \in \mathcal{A}$ and iteration t, the termination vectors are $V_i^{t-1}[k] = 0$ and $V_i^t[k] = 1$, and the termination iteration scalar is either $T^t > t$ or $T^t < t - D$, then there is a faulty termination status.

Proof. The proof of the first condition $T^t > t$ is trivial since it is always true that $T^t \leq t$ when agent i updates any entry of the termination vector. For the second condition, we use a similar proof to Proposition P3 to reach a contradiction. Assume $V_i^{t-1}[k] = 0$, $V_i^t[k] = 1$, and $T^t < t - D$. This implies that agent k satisfies the local termination criterion before iteration $t - D$. Assume agent k satisfies the local termination criterion at iteration t_k, i.e., $V_k^{t_k-1}[k] = 0$ and $B_k^{t_k} = 1$. Thus, $V_k^{t_k}[k] = 1$ and $T_k^{t_k} = t_k$ due to Rule R2. Similar to the proof of Proposition P3, we can show that $V_i^{t_k+d(k,i)-1}[k] = 0$, $V_i^{t_k+d(k,i)}[k] = 1$ and $T_i^{t_k+d(k,i)} \geq t_k$, for all $i \in \mathcal{A}$, where the inequality is due to the possibility that other agents may satisfy their local termination criteria after iteration t_k. This implies that, at iteration $t = t_k + d(k, i)$, the local termination statuses of agent i are $V_i^{t-1}[k] = 0$, $V_i^t[k] = 1$, and $T^t \geq t - d(k, i)$. Since $d(i, k) \leq D$, then $T^t \geq t-D$, which contradicts the assumption that $T^t < t-D$. Thus, there is a faulty termination status. □

Propositions P4 and P5 provide sufficient conditions to diagnose the correctness of the termination statuses. Proposition P4 detects discrepancies in the updates of the termination vectors, and Proposition P5 bounds the updates of the termination iteration scalar. We build on Propositions P4 and P5 to develop a fault-tolerant termination method that self-correct any faulty termination status. For the fault-tolerant method to be effective, we need two additional assumptions.

Assumption 4 (A4)**.** The faulty agents do not form a cut-set of the communication network.

A graph cut-set is any set $\mathcal{C} \subset \mathcal{A}$ such that removing the agents in $\mathcal{C}$ from the graph $G(\mathcal{A}, \mathcal{E})$ results in a disconnected subgraph. Assumption A4 ensures the existence of a path between the faulted agent and other agents without passing through the faulty agents. This path permits the correct termination statuses to traverse all agents. When the faulty agents form a cut-set, they divide the communication network into two parts, and there is no way for one part to know the correct termination statuses of the other part without passing the termination status through the faulty agents.

Assumption 5 (A5)**.** The distributed optimization computations are correct and accurate.

Although faults can occur in both the distributed optimization and the termination method, the proposed method focuses on faults in the termination method. We refer the reader to our other work [24]–[26] and the work in [27]–[31] on detecting data manipulation in distributed optimization algorithms. Assumption A5 implies that each agent shares the accurate status for its own local termination, but not necessarily the accurate termination statuses of other agents. In Section VI-C, we will show that this assumption can be relaxed to some extent while still ensuring that the fault-tolerant method will terminate the distributed algorithm appropriately.

A. Fault-Tolerant Method Procedure

The fault-tolerant extension incorporates the conditions in Propositions P4 and P5 to clear any faulty termination status. We modify the rules in Section III and also add three new rules to prevent faulty termination. In additional to the termination vector $V_i \in \{0, 1\}^{|\mathcal{A}|}$ and termination iteration scalar $T_i \in \mathcal{T}$, agents store a *termination iteration vector* $U_i \in \mathcal{T}^{|\mathcal{A}|}$ and a *correction vector* $C_i \in \mathbb{Z}_{\geq 0}^{|\mathcal{A}|}$. The termination iteration vector stores the iteration number when each agent satisfies its local termination criterion, and the correction vector tracks the number of iterations that are needed to reject any new update after a faulty status is detected.

Let t be the current iteration and B_i^t be the status of agent i's local termination criterion at iteration t. At iteration $t = 0$, agent i initializes the local termination status as follows:

Rule 0 (R0')**.** Set $V_i^0[k] = 0$, $U_i^0[k] = 0$, and $C_i^0[k] = 0$, for all $k \in \mathcal{A}$, and $T_i^0 = 0$.

At iteration $t \geq 1$, agent i receives the termination statuses V_j^{t-1}, U_j^{t-1}, and T_j^{t-1} from neighboring agents $j \in \mathcal{N}(i)$ and applies the following rules:

Rule 1 (R1')**.** If $B_i^t = 1$, $V_i^{t-1}[i] = 0$, and $C_i^{t-1}[i] = 0$, then set $V_i^t[i] = 1$, and $U_i^t[i] = t$.

Rule 2 (R2')**.** For each $k \in \mathcal{A} \setminus \{i\}$, let $\mathcal{J}_k = \left\{ j \mid j \in \mathcal{N}(i), V_j^{t-1}[k] = 1, U_j^{t-1}[k] \in [t - D, t - 1] \right\}$.

If $V_i^{t-1}[k]=0$, $C_i^{t-1}[k]=0$, and $\mathcal{J}_k \neq \emptyset$, then set $V_i^t[k]=1$ and $U_i^t[k]=\max_{j\in\mathcal{J}_k}\left\{U_j^{t-1}[k]\right\}$.

Rule 3 (R3'). Set $T_i^t=\max_{k\in\mathcal{A}}\left\{U_i^t[k]\right\}$.

At iteration $t\geq 2$, in addition to Rules R1'–R3', agent i applies the following rules:

Rule 4 (R4'). For each $k\in\mathcal{A}$, if the termination vectors $V_i^{t-2}[k]=1$ and $V_i^{t-1}[k]=1$, and there is $V_j^{t-1}[k]=0$ or $V_j^{t-1}[k]=1$ with $U_j^{t-1}[k]\neq U_i^{t-1}[k]$ for any $j\in\mathcal{N}(i)$, then set $V_i^t[k]=0$, $U_i^t[k]=t$, $T_i^t=t$, and $C_i^t[k]=U_i^{t-1}[k]+2D+|\mathcal{A}|-1-t$.

Rule 5 (R5'). For each $k\in\mathcal{A}$, if $C_i^{t-1}[k]>0$, then set $C_i^t[k]=C_i^{t-1}[k]-1$.

Rule 6 (R6'). If $\sum_{k\in\mathcal{A}}V_i^t[k]=|\mathcal{A}|$ and $t\geq T_i^t+2D+|\mathcal{A}|-1$, terminate.

These rules are similar to the termination method in Section III with additional conditions that mandate clearing any faulty termination statuses. The main differences here are: 1) only accepting updates that do not violate the conditions in Proposition P5 as stated in Rule R2', 2) clearing any status that violates the conditions in Proposition P4 as stated in Rule R4', 3) after a fault is detected, rejecting any new update for a certain number of iterations computed by Rule R4' and updated by Rule R5', and 4) accounting for the passing of the correction termination status from the faulted agent to the other agents in the termination conditions as stated in Rule R6'.

Selecting the number of iterations to reject new updates plays an important role in the correctness of the fault-tolerant method as we will show in the correctness proofs. When an agent detects a fault via Proposition P4, the agent keeps the termination status vector equal to zero for the next $C_i^t[k]=U_i^{t-1}[k]+D+|\mathcal{A}|-1-t$ iterations due to Rules R4' and R5'. During these iterations, agents clear the faulty termination status before allowing any change to the termination status. The use of the iteration vector $U_i^{t-1}[k]$ ensures that all agents use the same iteration number of the faulty status to compute $C_i^t[k]$. Thus, the value of $C_i^t[k]$ will be exactly the same for all agents that receive the same faulty status.

The fault-tolerant method requires storing additional information from the previous iterations. Specifically, each agent needs to store the termination vectors from the last two iterations ($V_i\in\{0,1\}^{|\mathcal{A}|}$), the termination iteration vector ($U_i\in\mathbb{Z}_{\geq 0}^{|\mathcal{A}|}$), the termination iteration scalar ($T_i\in\mathbb{Z}_{\geq 0}$), and the correction vector ($C_i\in\mathbb{Z}_{\geq 0}^{|\mathcal{A}|}$). Thus, the total stored data for agent i is $2|\mathcal{A}|$ binaries and $2|\mathcal{A}|+1$ non-negative integers.

B. Fault-Tolerant Method Correctness

Since Proposition P1 does not hold when there is a faulty termination status, we can not use Proposition P3 to prove that the algorithm will terminate appropriately. We will instead use the following three propositions to show the correctness of the fault-tolerant extension. We first bound the maximum number of iterations for any faulty status to persist in the termination vectors. Then, we show that no faulty status can terminate the algorithms. Lastly, we show that the algorithm terminates appropriately if there are no new fault injections.

Proposition 6 (P6). The maximum number of iterations that any faulty termination status will persist in any termination vector (V_i, for any $i\in\mathcal{A}$) is at most $D+|\mathcal{A}|-2$ iterations.

We prove Proposition P6 by tracing the path of the faulty termination status from the faulty agent to the faulted agent passing through an arbitrary agent using Rule R2', and then tracing back the correction status from the faulted agent to the same arbitrary agent using Rule R4'.

Proof. Without loss of generality, assume there is a faulty agent j, a faulted agent k ($j\neq k$ due to Assumption A5), and the fault is injected at iteration t_j. Consider any arbitrary agent i that receives the faulty status from agent j at iteration t_i and let $M=t_i-t_j$, i.e., the length of the shortest path between j and i such that the faulted agent k is not in that path. We want to show that this faulty status will be cleared at iteration $t_i'\leq t_i+D+|\mathcal{A}|-2$.

Let $P(i,k)=\{i=i_0,i_1,\ldots,i_r,\ldots,i_R=k\}$ be the shortest path between i and k with length R such that $j\notin P(i,k)$ (this path exists due to Assumption A4). Agents along this path either receive or do not receive the faulty status at iteration $t_i=t_j+M$. At the next S iterations, one or more agents in the path $P(i,k)$ will receive the faulty status for the first time at each iteration. At iteration t_i+s, where $s\leq S$, we can trace the agents that receive the faulty status along the path $P(i,k)$ starting from agent i until some agent $i_r\in P(i,k)$ that did not receive the faulty status yet. The next iteration t_i+s+1, agent i_r will either receive the faulty status or reject the faulty status. The rejection could happen because one of two possibilities: 1) $M+s=D+1$ due to the condition in Rule R2' (reject any new update that takes more than D iterations) or 2) agent i_r received and cleared the faulty status before iteration t_i+s and $C_{i_r}^{t_i+s}[k]>0$ (if $C_{i_r}^{t_i+s}[k]>0$ agent i_r rejects any new update due to Rule R2'). Eventually, there will be an agent $i_r\in P(i,k)$ that will reject the faulty status update at iteration S with $M+S\leq D+1$ since the faulted agent $k\in P(i,k)$.

This implies the termination vector of agent $i_{r-1}\in P(i,k)$ (i.e., i_{r-1} is the neighbor of i_r in the path $P(i,k)$ with shortest distance to agent i) is $V_{i_{r-1}}^{t_i+S+1}[k]=0$ due to Rule R4'. The next iterations, agents along the path $P(i,i_r)\subseteq P(i,k)$ will also clear the faulty status leading to $V_{i_0}^{t_i+S+r}[k]=0$ due to Rule R4'. This implies agent i will clear the faulty status at iteration $t_i'=t_i+S+r\leq t_i+D+|\mathcal{A}|-2$ since $S\leq D+1-M$, $M\geq 1$, and $r\leq R\leq|\mathcal{A}|-2$. Thus, the maximum number of iterations for any faulty status to persist in agent i's termination vector is $t_i'-t_i\leq D+|\mathcal{A}|-2$. □

We can use the same proof to show that the same bound also applies when there are multiple faulty agents. With multiple faulty agents, we need to differentiate between two cases. If the faulty agents collude in the sense that they use the same value for the faulty termination iteration vector $U_j[k]$, then the exact same proof applies. On the other hand, if the faulty agents use different values for the faulty termination iteration vector $U_j[k]$, then when following the path $P(i,k)$ in the proof, an agent $i_{r-1}\in P(i,k)$ might clear the faulty status because

of the discrepancy in the values of the termination iteration vectors with agent $i_r \in P(i,k)$ due to Rule R4'. Note that the bound in the proposition is the tightest bound for any general graph as shown in the example in Figure 1. Next, we use Proposition P6 to show that no faulty status can terminate the distributed algorithm.

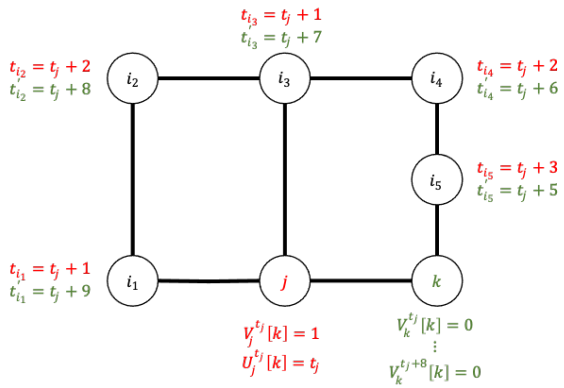

Figure 1. An example for a faulty status where the bound in Proposition P6 is tight. The number of agents $|\mathcal{A}| = 7$ and the communication network diameter $D = 3$. The fault originates from agent j (in red) at iteration t_j and the faulted agent is k. At iteration $t_j + 3$, all agents receive the faulty status (in red). Agent i_5 is the first agent that clears the faulty status since agent i_5 is a neighbor to the faulted agent k. Agents clear the faulty status at iteration t_i' (in green). At iteration $t_j + 9$, agent i_1 is the last agent to clear the faulty status. Thus, the number of iterations agent i_1 needs to clear the faulty status is $D + |\mathcal{A}| - 2 = 8$ iterations.

Proposition 7 (P7)**.** No agent will terminate the computations before satisfying the global termination criterion.

The proof of Proposition P7 is a direct consequence of Proposition P6. We only need to show that any fault will be cleared before the termination condition in Rule R6' is true.

Proof. The only way for any agent to terminate the computation before satisfying the global termination criterion is to have a faulty status in the termination vector. Assume agent i receives a faulty termination status at iteration t_i. Since the maximum number of iterations for any faulty status to persist in the termination vectors is $D + |\mathcal{A}| - 2$ as stated in Proposition P6, the iteration t_i' at which agent i clears the faulty status must be no later than iteration $t_i + D + |\mathcal{A}| - 2$. From Rules R2' and R3', we know that $T_i^{t_i} \geq t_i - D$, otherwise agent i will not receive the faulty status due to Rule R2'. Thus, agent i terminates using Rule R6' by at least iteration $t \geq T_i^{t_i} + 2D + |\mathcal{A}| - 1 \geq t_i + D + |\mathcal{A}| - 1$. However, since agent i clears the faulty status at iteration $t_i' \leq t_i + D + |\mathcal{A}| - 2 < t_i + D + |\mathcal{A}| - 1 \leq t$, agent i will not terminate by any faulty termination status. Thus, the algorithm will not terminate before satisfying the global termination criterion. □

When a faulty agent keeps injecting faulty termination status, the algorithm will not terminate, as implied by Proposition P7. However, if the faulty agent stops injecting faulty termination statuses, Rules R4' and R5' ensure clearing the faulty termination status in all agents' local termination vectors (V_i, for all $i \in \mathcal{A}$). If all agents clear the faulty termination status, then Propositions P1 and P2 are valid, and the algorithm terminates appropriately using Rule R6'. The last proposition establishes that Rules R4' and R5' clear the faulty termination status if there is no new fault injections.

Proposition 8 (P8)**.** If the distributed algorithm satisfies the global termination criterion at iteration T_G and there is no fault injections at any iteration $t \geq T_G$, then the algorithm will terminate appropriately.

We prove Proposition P8 by showing that whenever there is a faulty status, there will be a future iteration where all agents clear that faulty status, and this iteration must be before any agent terminates the computation due to the faulty status. We achieve this result using the correction vector C_i, which prevents agents from accepting any new updates until waiting a certain number of iterations after detecting a faulty status.

Proof. Without loss of generality, assume there is a fault injection by a faulty agent j and a faulted agent k ($k \neq j$ due to Assumption A5). Further, assume the faulted agent is the last agent to satisfy its local termination criterion and the faulty agent stops injecting the faulty status before the faulted agent k satisfies its local termination criterion. Denote the k-th entry of the faulty termination iteration vector as $U_j[k] = t_j$. For any arbitrary agent i that receives the faulty status at iteration t_i, the local termination statuses are $V_i^{t_i}[k] = 1$, $U_i^{t_i}[k] = t_j$, and $t_i \leq t_j + D$ due to Rule R2'.

From Proposition P6, agent i clears the faulty status by iteration $t_i' \leq t_i + D + |\mathcal{A}| - 2$. The local termination statuses at this iteration are $V_i^{t_i'}[k] = 0$ and $C_i^{t_i'}[k] = U_i^{t_i} + 2D + |\mathcal{A}| - 1 - t_i' = t_j + 2D + |\mathcal{A}| - 1 - t_i'$. Since $t_i' \leq t_i + D + |\mathcal{A}| - 2$ and $t_i \leq t_j + D$, then $t_i' \leq t_j + 2D + |\mathcal{A}| - 2$. This implies $C_i^{t_i'}[k] \geq t_j + 2D + |\mathcal{A}| - 1 - (t_j + 2D + |\mathcal{A}| - 2) = 1 > 0$. Since $C_i^{t_i'}[k] > 0$, agent i will not update its local termination status during the next $C_i^{t_i'}[k]$ iterations. Let t'' be the last iteration where agent i will not update its termination status. We have $t'' = t_i' + C_i^{t_i'}[k] = t_j + 2D + |\mathcal{A}| - 1$. Since t'' only depends on the faulty status, then the termination vector of any agent $i \in \mathcal{A} \setminus \{j\}$ that receives the faulty status is $V_i^{t''}[k] = 0$. Thus, all agents clear the faulty status and do not accept any updates until iteration t'' if the faulted agent k does not satisfy its termination criterion.

Let t_k be the iteration when the faulted agent k satisfies its local termination criterion. If $t_k \geq t''$, then we can use Propositions P1–P3 to show that the algorithm terminates appropriately. We still need to show that the algorithm terminates appropriately if $t_k < t''$.

If $t_k < t''$, we can partition the agents into two sets based on whether they received the correct termination status. Let $\mathcal{I} = \{i \mid i \in \mathcal{A}, V_i^{t_i}[k] = 1, U_i^{t_i}[k] = t_k, t_i \leq t_k + D\}$ be the set of agents that receive the correct termination status before iteration $t_k + D$, and let $\mathcal{I}^c = \mathcal{A} \setminus \mathcal{I}$ be the remaining agents.

If $\mathcal{I}^c = \emptyset$, then we can use Propositions P1–P3 to show that the algorithm terminates appropriately. On the other hand, if $\mathcal{I}^c \neq \emptyset$, then we know that there is an iteration t'' where $V_i^{t''}[k] = 0$, for all $i \in \mathcal{I}^c$. Since $\mathcal{I}$ and $\mathcal{I}^c$ form a graph partition, we always have some agent $i \in \mathcal{N}(i')$ where $i \in \mathcal{I}$

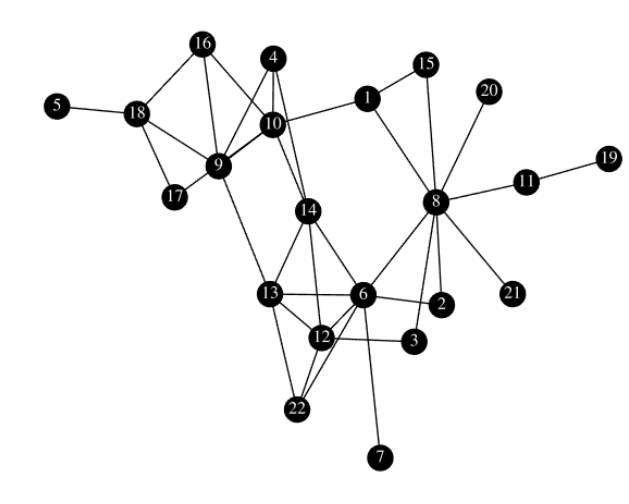

Figure 2. Communication network of the 240-bus power system with 22 agents. The nodes are the agents and the edges are the communication links.

and $i' \in \mathcal{I}^{\mathrm{c}}$. Thus, agents in $\mathcal{I}$ will also clear the correct termination status due to Rule R4'. Moreover, those agents will not receive any new updates until some iteration $t''_k = t_k + 2D + |\mathcal{A}| - 1$. Since $t_k > t_j$ by assumption, then $t''_k > t''$. Thus, we have $V_i^{t''_k}[k] = 0$, for all $i \in \mathcal{A}$. We can then use Propositions P1–P3 to show that the algorithm terminates appropriately in this case. □

The same proof applies when there are multiple faulty agents. The only restrictions are that the faulty agents do not form a cut-set as Assumption A4 states, and the faulty agents do not terminate the computation before satisfying their local termination criterion as Assumption A5 implies.

V. Simulation Results

To demonstrate the proposed termination method, we use the Alternating Direction Method of Multipliers (ADMM) algorithm to solve the Optimal Power Flow problem (OPF) problem. OPF problems are used to optimize the dispatch of generators in electric power grids. We consider a 240-bus test case with 22 areas from PGLib-OPF benchmark library [32]. Each agent controls an area and communicates with neighboring agents when there is an electric transmission line connecting the two areas. As shown in Figure 2, the communication network consists of 36 links and has a diameter $D = 7$. We decompose the power system into multiple subsystems using the method in [33] and, for illustrative purposes, solve the DC approximation of the OPF problem. For the local termination criterion, we use the shared variables' mismatches (i.e., the differences in the voltage phase angles of the shared buses between neighboring agents) having an ℓ_∞-norm less than 10^{-2} radians.

The agents satisfy the global termination criterion for the first time after 862 iterations. Agent 8 is the last agent to satisfy its local termination criterion at this iteration. The algorithm then terminates using the proposed method at iteration 869, seven iterations after the the global termination criterion is satisfied. Figure 3 shows the agents' termination statuses for the last nine iterations. Information regarding satisfaction of the global termination status takes five iterations to traverse all the agents. However, based on the diameter of the communication network ($D = 7$), the agents perform two additional iterations before terminating to ensure that all the agents are informed about satisfaction of the global termination criterion.

To demonstrate the fault-tolerant extension of the termination method described in Section IV, we next consider a case where some agents inject faulty termination statuses. Faulty agents inject a faulty termination every 100 iterations by setting $V_f^{t_f}[k] = 1$, for all $k \in \mathcal{A}$, $f \in \mathcal{F}$, and $t_f \in \{100, 200, 300, 400, 500, 600, 700, 800\}$, where $\mathcal{F}$ is the set of faulty agents. Moreover, we have the faulty agents send faulty status for 20 iterations. To maximize the possibility of terminating the algorithm, we set the value of the termination iteration vector to be $U_f^{t_f}[k] = U_f^{t_f-1}[k]$, for all $k \in \mathcal{A}$, if $V_f^{t_f-1}[k] = 1$ and $U_f^{t_f}[k] = t_f$ otherwise. We stop injecting the faulty termination status after iteration 820 to verify that the algorithm will then terminate appropriately.

Table I summarizes the simulations for cases with one to five faulty agents. The distributed algorithm terminates in 897 iterations in all cases. The maximum number of iterations that the agents presume the global termination criterion is satisfied because of the faulty termination status is seven iterations. Furthermore, the maximum number of iterations for any single fault to persist in any local termination vector other than the faulty agent is eight iterations. Despite the presence of faulty agents, the algorithm terminates appropriately in all cases.

Figure 4 shows the number of agents that satisfy the local and the global termination criteria when five faulty agents collude in an attempt to terminate the computation before the global termination criterion is satisfied. Observe that the number of agents that receive the faulty global termination criterion spikes after any faulty status is injected by the faulty agents and returns to zero after a few iterations.

Table I
Faulty Test Case Results Summary

Faulty Agents	Termination Iteration	Maximum Single Fault Persists	Maximum Global Fault Persists
1	897	7	6
1, 3	897	7	5
1, 3, 4	897	7	5
1, 3, 4, 9	897	8	7
1, 3, 4, 9, 14	897	8	7

VI. Remarks and Extensions

This section highlights several remarks and introduces extensions of the proposed method to reduce the computational burden, detect faulty agents, revise assumptions, and terminate asynchronous distributed optimization algorithms.

A. Reducing Additional Iterations

The proposed method in Section III terminates distributed optimization algorithms after D iterations of satisfying the global termination criterion to ensure that the correct termination status is communicated to all agents. Moreover, the fault-tolerant termination method in Section IV uses an additional

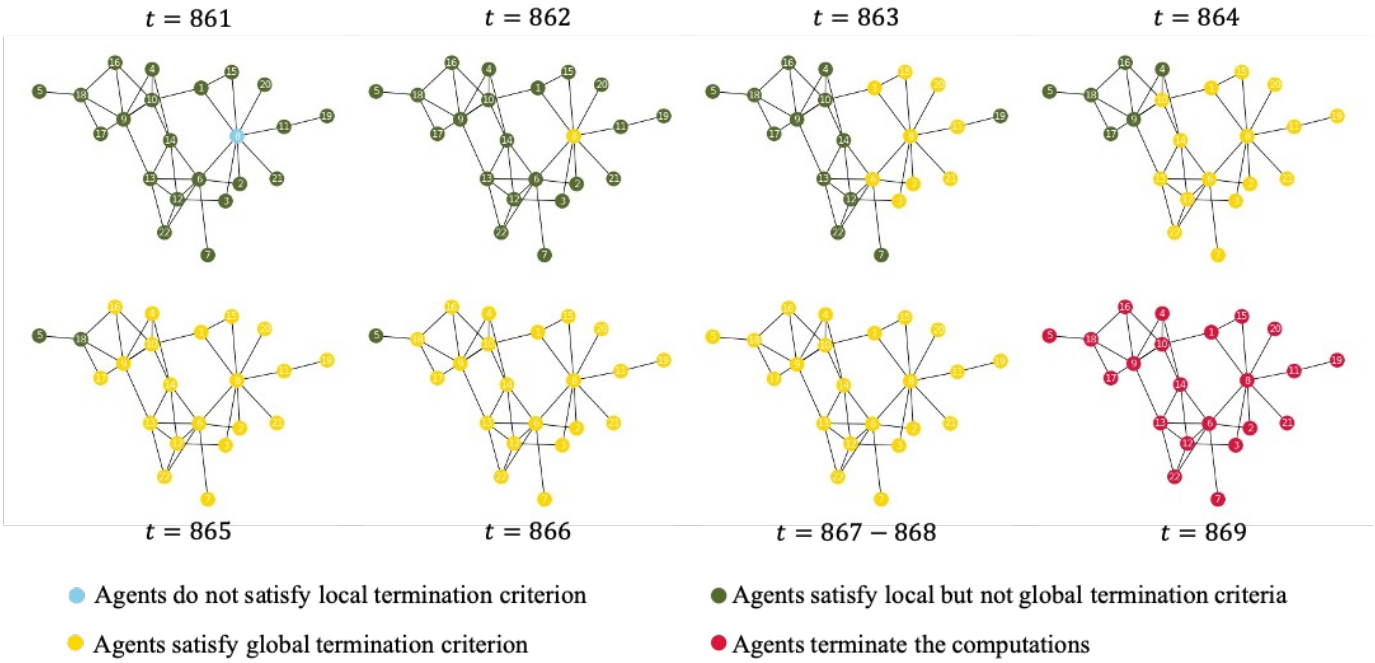

Figure 3. Termination status for the last nine iterations of the 240-bus optimal power flow test case solved using the alternating direction method of multipliers and the proposed termination method. At iteration $t = 861$, all agents satisfy their local termination criterion (green nodes) expect agent 8 (cyan node). Agent 8 satisfies its local termination criterion at iteration $t = 862$ and acknowledges the global termination criterion (becomes yellow). During iterations $t = 863$ – 868, the global termination criterion then traverses the other agents (they become yellow). In iterations $t = 867$ – 868, all agents have received the global termination criterion (all nodes are yellow) but the agents do not terminate the computation since they are not yet assured that the global termination status information has reached all agents. This occurs at iteration $t = 869$, at which point all agents terminate the computation appropriately (red nodes).

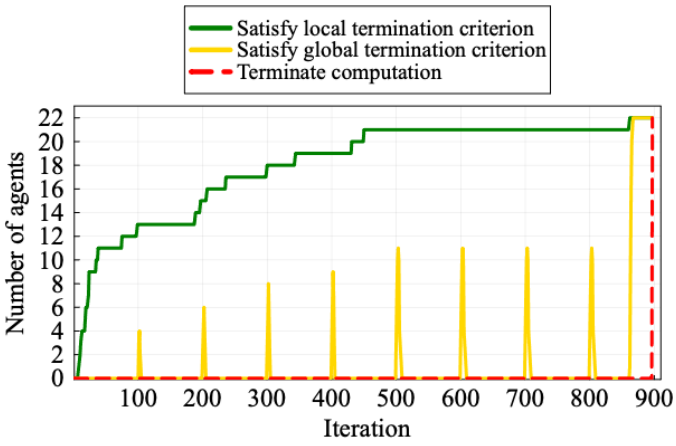

Figure 4. Number of agents that satisfy the local (green) and global (yellow) termination criteria and terminate computation (red) when five faulty agents inject faulty statuses for the first 20 of every 100 iterations, stopping at iteration 820. The algorithm then appropriately terminates at iteration 897.

$2D+|\mathcal{A}|-1$ iterations to prevent faulty early termination. The additional $D + |\mathcal{A}| - 1$ iterations are thus a trade-off between reliability and computational burden.

We can reduce the number of computational iterations per agent while maintaining the fault tolerance guarantees. Rather than performing the local computations after an agent satisfies the global termination criterion, the agent can stop performing the normal computations and instead only communicate the termination status. In this case, once all agents are assured to have received the global termination status, the agents terminate and use the solution from iteration T_G instead of the last iteration's solution. If there are any discrepancies in the termination statuses due to a faulty agent, the agents continue from where they stopped.

This modification will reduce the number of additional computations needed in the fault-tolerant method by more than 66%. This reduction comes from the fact that all agents receive the global termination status in a maximum of D iterations. In the remaining $D + |\mathcal{A}| - 1$ iterations, the agents participate in the termination method only, which has a negligible computational burden compared with solving the agent's local optimization problem. Thus, the fraction of avoided additional computations is $\frac{D+|\mathcal{A}|-1}{2D+|\mathcal{A}|-1}$, which is lower bounded by 0.66.

B. Detecting Faulty Agents

The fault-tolerant method self-corrects any faulty status but does not provide information about the faulty agents. If faulty agents continue sending a faulty termination status, we can show that the faulty agents can be detected by their neighbors. Specifically, for some iteration t and agent i, if $V_i^{t-2}[k] = 1$, $V_i^{t-1}[k] = 0$, and $V_j^t[k] = 1$ for some neighbor $j \in N(i)$, then j is a faulty agent since it does not follow Rule R4'. In other words, if an agent clears a fault at a certain iteration, and there are one or more neighbors that do not clear the same fault in the subsequent iteration, then those neighbors are faulty agents.

C. Revising Assumptions

To prove the termination method's correctness, we use Assumption A3 which states that the termination criterion is monotone. However, we can relax this assumption in the fault-tolerant method. We can treat any non-monotone behavior of the local termination criterion status as a fault injection. The fault-tolerant method will then self-correct the status to reflect the change in the non-monotone local termination criterion status. The only required assumption for the fault-tolerant method with a non-monotone local termination criterion is that if the local termination criterion is satisfied for $D + |\mathcal{A}| - 1$ iterations then it remains satisfied for all subsequent iterations, which is weaker than Assumption A3.

As stated in Assumption A4, the fault-tolerant method requires having a path between any two agents that does not pass through a faulty agent. This suggests that the best way to prevent faulty termination when using the proposed method is to increase the minimum number of agents in any cut-set by increasing the communication network's connectivity. If the communication network is k-connected, i.e., there are k disjoint paths between any pairs of agents, then we are assured that the distributed algorithm will avoid faulty termination with up to $k - 1$ faulty agents.

Assumption A5 isolates faults occurring in the termination method from faults in the distributed optimization algorithm itself. Although we use Assumption A5 to prove the correctness of the fault-tolerant method, we can relax this assumption. The proofs of Propositions P6–P8 assume the faulty and faulted agents are not the same agent. However, the only assumption we need in these proofs is that the faulty agent is not the last agent that satisfies the local termination criterion.

D. Asynchronous Distributed Optimization Algorithms

The proposed termination method is presented in the framework of synchronized distributed optimization where all agents communicate at common intervals. However, the method can be modified to terminate asynchronous distributed optimization algorithms. The modification requires a synchronized time measurement that records the time at which the last local termination is satisfied. This time measurement replaces the termination iteration scalar (T_i, for all $i \in \mathcal{A}$). Furthermore, the agents need to store the local solutions with a time stamp for each solution at each local iteration.

Asynchronous distributed algorithms can be appropriately terminated as follows. When an agent satisfies the local termination criterion, the agent records the time in T_i and shares the termination status (T_i and V_i) with the neighboring agents. Agents then use Rule R2 to update the termination vector (V_i) and update the termination time (T_i) similar to the termination iteration scalar. Lastly, agents use the first condition in Rule R3 to terminate, that is, terminate if $\sum_{k\in\mathcal{A}} V_i^t[k] = |\mathcal{A}|$. The solution of the distributed algorithm in this case is the last local solution with a timestamp equal to or before the time in the local termination status (T_i). We note that this extension works for asynchronous distributed optimization termination without faulty agents, but further work is needed to extend the fault-tolerant method to asynchronous settings.

VII. Conclusion

This paper proposes a method for terminating distributed optimization algorithms solved by multiple computing agents. The proposed method uses three simple rules for each agent that solely rely on local information. Furthermore, we propose a fault-tolerant extension to the method via three additional rules. These rules are symmetrical (same for every agent), and no central entity is required to terminate the distributed optimization algorithm. Further, the proposed method requires minimal knowledge of the global system. The only prior information that we assume the agents know are the number of agents and the diameter of the communication network.

There are several directions for future work that build on the method proposed in this paper. First, we note that proving the method's correctness requires the local termination criterion to be monotone, i.e., once satisfied, each agent's local termination status remains satisfied for all future iterations. However, the local termination criterion may not be monotone for some distributed optimization algorithms. We therefore plan to analyze local termination criterion satisfaction for various distributed optimization algorithms and design alternate criteria that exhibit monotone behavior. Second, our contemporaneous work in [26] studies faults in the distributed optimization algorithm's computations. By combining ideas in [26] with the results of this paper, we aim to simultaneously address faults in both the distributed optimization algorithm's computations and the termination method. Third, we plan to extend the fault-tolerant method to consider asynchronous distributed optimization as well as dynamic communication networks where connections between agents may vary across iterations.

Acknowledgment

The authors would like to thank Rachel Harris for insightful discussions and feedback on the paper.

References

[1] G. Notarstefano, I. Notarnicola, and A. Camisa, "Distributed optimization for smart cyber-physical networks," *Foundations and Trends® in Systems and Control*, vol. 7, no. 3, pp. 253–383, 2019.

[2] D. Molzahn, F. Dörfler, H. Sandberg, S. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, "A survey of distributed optimization and control algorithms for electric power systems," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017.

[3] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, 2004, pp. 20–27.

[4] T. Halsted, O. Shorinwa, J. Yu, and M. Schwager, "A survey of distributed optimization methods for multi-robot systems," arXiv:2103.12840, 2021.

[5] S. Hu, X. Chen, W. Ni, E. Hossain, and X. Wang, "Distributed machine learning for wireless communication networks: Techniques, architectures, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1458–1493, 2021.

[6] W. Fokkink, *Distributed Algorithms: An Intuitive Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[7] N. Francez, "Distributed termination," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 2, no. 1, pp. 42–55, 1980.

[8] E. W. Dijkstra, "Termination detection for diffusing computations," *Information Processing Letters*, vol. 11, no. 1, pp. 1–4, 1980.

[9] J. Matocha and T. Camp, "A taxonomy of distributed termination detection algorithms," *Journal of Systems and Software*, vol. 43, no. 3, pp. 207–221, 1998.

[10] F. Mattern, "Global quiescence detection based on credit distribution and recovery," *Information Processing Letters*, vol. 30, no. 4, pp. 195–200, 1989.

[11] S. Rana, "A distributed solution of the distributed termination problem," *Information Processing Letters*, vol. 17, no. 1, pp. 43–46, 1983.

[12] W. H. J. Feijen and A. J. M. Gasteren, "Shmuel Safra's termination detection algorithm," in *On a Method of Multiprogramming*. New York, NY, USA: Springer, 1999, ch. 29, pp. 313–332.

[13] N. Shavit and N. Francez, "A new approach to detection of locally indicative stability," in *Proceedings of the International Colloquium on Automata, Languages, and Programming*, 1986.

[14] R. F. DeMara, Y. Tseng, and A. Ejnioui, "Tiered algorithm for distributed process quiescence and termination detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1529–1538, 2007.

[15] F. Mattern, "Experience with a new distributed termination detection algorithm," in *Proceedings of the 2nd International Workshop on Distributed Algorithms*, 1987, pp. 127–143.

[16] J. Misra, "Detecting termination of distributed computations using markers," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983, pp. 290–294.

[17] T. Lai and L. Wu, "An (N-1)-resilient algorithm for distributed termination detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 1, pp. 63–78, 1995.

[18] G. Karlos, W. Fokkink, and P. Fuchs, "Fault-tolerant termination detection with Safra's algorithm," in *Proceedings of the International Conference on Networked Systems*, 2021, pp. 71–87.

[19] A. A. Khokhar, S. E. Hambrusch, and E. Kocalar, "Termination detection in data-driven parallel computations/applications," *Journal of Parallel and Distributed Computing*, vol. 63, no. 3, pp. 312–326, 2003.

[20] A. H. Baker, S. Crivelli, and E. R. Jessup, "An efficient parallel termination detection algorithm," *The International Journal of Parallel, Emergent and Distributed Systems*, vol. 21, no. 4, pp. 293–301, 2006.

[21] B. Jin, H. Li, W. Yan, and M. Cao, "Distributed model predictive control and optimization for linear systems with global constraints and time-varying communication," *IEEE Transactions on Automatic Control*, vol. 66, no. 7, pp. 3393–3400, 2021.

[22] F. Bénézit, V. Blondel, P. Thiran, J. Tsitsiklis, and M. Vetterli, "Weighted gossip: Distributed averaging using non-doubly stochastic matrices," in *Proceedings of the IEEE International Symposium on Information Theory*, 2010, pp. 1753–1757.

[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[24] M. Alkhraijah, R. Harris, S. Litchfield, D. Huggins, and D. K. Molzahn, "Analyzing malicious data injection attacks on distributed optimal power flow algorithms," in *54th North American Power Symposium (NAPS)*, 2022.

[25] R. Harris and D. K. Molzahn, "Detecting and mitigating data integrity attacks on distributed algorithms for optimal power flow using machine learning," *57th Hawaii International Conference on System Sciences (HICSS)*, 2024.

[26] M. Alkhraijah, R. Harris, S. Litchfield, D. Huggins, and D. K. Molzahn, "Detecting shared data manipulation in distributed optimization algorithms," arXiv:2310.13252, 2023.

[27] E. Munsing and S. Moura, "Cybersecurity in distributed and fully-decentralized optimization: Distortions, noise injection, and ADMM," arXiv:1805.11194, 2018.

[28] L. Majzoobi, F. Lahouti, and V. Shah-Mansouri, "Analysis of distributed ADMM algorithm for consensus optimization in presence of node error," *IEEE Transactions on Signal Processing*, vol. 67, no. 7, pp. 1774–1784, 2019.

[29] Z. Cheng and M. Chow, "Resilient collaborative distributed AC optimal power flow against false data injection attacks: A theoretical framework," *IEEE Transactions on Smart Grid*, vol. 13, no. 1, pp. 795–806, 2022.

[30] Y. Yang, G. Raman, J. Peng, and Z. Ye, "Resilient consensus-based AC optimal power flow against data integrity attacks using PLC," *IEEE Transactions on Smart Grid*, vol. 13, no. 5, pp. 3786–3797, 2022.

[31] A. Gusrialdi and Z. Qu, "Resilient distributed optimization against cyber-attacks," *IEEE Control Systems Letters*, vol. 7, pp. 3956–3961, 2023.

[32] IEEE PES Task Force on Benchmarks for Validation of Emerging Power System Algorithms, "The Power Grid Library for benchmarking AC optimal power flow algorithms," arXiv:1908.02788, 2019.

[33] M. Alkhraijah, R. Harris, C. Coffrin, and D. K. Molzahn, "PowerModelsADA: A framework for solving optimal power flow using distributed algorithms," *IEEE Transactions on Power Systems*, vol. 39, no. 1, pp. 2357–2360, 2024.