Team Formation and Applications

Yuval Emek
Technion, Haifa, Israel

Shay Kutten[1]
Technion, Haifa, Israel
GTEP, Haifa, Israel

Ido Rafael
Technion, Haifa, Israel

Gadi Taubenfeld
Reichman University, Herzliya, Israel

Abstract

A novel long-lived distributed problem, called *Team Formation (TF)*, is introduced together with a message- and time-efficient randomized algorithm. The problem is defined over the asynchronous model with a complete communication graph, using bounded size messages, where a certain fraction of the nodes may experience a generalized, strictly stronger, version of initial failures. The goal of a TF algorithm is to assemble tokens injected by the environment, in a distributed manner, into *teams* of size σ, where σ is a parameter of the problem.

The usefulness of TF is demonstrated by using it to derive efficient algorithms for many distributed problems. Specifically, we show that various (one-shot as well as long-lived) distributed problems reduce to TF. This includes well-known (and extensively studied) distributed problems such as several versions of leader election and threshold detection. For example, we are the first to break the linear message complexity bound for asynchronous implicit leader election. We also improve the time complexity of message-optimal algorithms for asynchronous explicit leader election. Other distributed problems that reduce to TF are new ones, including matching players in online gaming platforms, a generalization of gathering, constructing a perfect matching in an induced subgraph of the complete graph, and more. To complement our positive contribution, we establish a tight lower bound on the message complexity of TF algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases asynchronous message-passing, complete communication graph, initial failures, leader election, matching

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.30

Related Version A full version of this extended abstract is available, containing details omitted here. *Full Version*: `https://arxiv.org/abs/2508.13084` [22]

Funding *Yuval Emek*: Partially supported by an Israel Science Foundation (ISF) grant 730/24 and by the Grand Technion Energy Program (GTEP).
Shay Kutten: Partially supported by ISF grant 1346/22; his research was carried out within the framework of the Grand Technion Energy Program (GTEP).

Acknowledgements We would like to thank Eviatar Procaccia for his help with the proof of Lemma 9.

1 Introduction

Consider the following three problems, defined over an asynchronous message-passing system with a complete communication graph, where a constant fraction of the nodes may be faulty: (P1) electing a leader among the non-faulty nodes; (P2) constructing a perfect matching in

[1] a part of the work was done while visiting Fraunhofer SIT

the subgraph induced by a subset of the non-faulty nodes, specified (distributively) by the input; (P3) assembling Dungeons & Dragons parties from characters that arrive over time at the non-faulty nodes so that each party includes 3 wizards, 2 paladins, 2 rogues, and 1 monk. On the face of it, problems (P1)–(P3) have little in common. In particular, (P1) and (P2) are one-shot problems, whereas (P3) is long-lived; (P1) requires global symmetry breaking, whereas in (P2) and (P3) the symmetry breaking is local in essence. Therefore, it may come as a surprise that all three problems reduce to a single new problem.

In this paper, we introduce the aforementioned new problem, called *team formation*, and develop an efficient algorithm for it. Following that, we show how our algorithm leads to algorithmic solutions for various other problems (including (P1)–(P3)), improving their state-of-the-art. For example, we are the first to break the linear message complexity bound for asynchronous implicit leader election, see Sec. A.1.1. We also improve the time complexity of message-optimal algorithms for asynchronous explicit leader election, see Sec. A.1.2. On the negative side, we establish lower bounds on the communication demands of TF algorithms, see the full version [22].

1.1 The Basic Setting and Problem Definition

Consider an *asynchronous* message-passing system that consists of n nodes (a.k.a. processors). The communication structure is assumed to be a complete undirected graph over the node set V so that every two nodes may exchange messages with each other. These messages carry $O(\log n)$ bits of information (cf. the CONGEST model [52]) and are delivered with a finite delay, where the delay of each message is determined (individually) by an adversary. Unless stated otherwise, it is assumed that the nodes are anonymous and that each node distinguishes between its $n-1$ neighbors by means of (locally) unique *port numbers* [6, 61, 31]. The adversary may cause some nodes to become faulty, preventing them from participating in the execution in accordance with a generalization of the *initial failures* model [18, 24, 60], as long as a sufficiently large fraction of the nodes remain non-faulty throughout the execution.[2] A precise definition of the computational model is presented in Sec. 3.1.

Team Formation. We introduce a long-lived distributed problem called *team formation (TF)*, defined over an integral *team size* parameter $2 \leq \sigma \leq n$.[3] In a TF instance, abstract *tokens* are injected into the nodes over time, where the adversary determines the timing and location of these token injections. Tokens held by a node $v \in V$ can be transported to a node $v' \in V$ over a message sent from v to v'. Each token remains in the system until it is deleted, as explained below.

The correctness criterion of the TF problem is captured by the following two conditions:

Safety: a token can be deleted from the system only as part of a *team* that consists of exactly σ tokens, all of which are held by the same node and deleted simultaneously; the operation of deleting a team of tokens is referred to as *forming a team*.

Liveness: if the system contains at least σ tokens, then a team must be formed in a finite time.

[2] The failure model considered in this paper is identical to the classic model of initial failures when restricted to one-shot problems. As explained in the sequel, the generalization comes into play only in the context of long-lived problems.

[3] The TF problem is well defined also for $\sigma > n$, however, the restriction to $\sigma \leq n$ simplifies the discussion and is consistent with the applications presented in Sec. 1.4.

Assume without loss of generality that the system contains at most $n^{O(1)}$ tokens at any given time.[4] The tokens are assumed to be indistinguishable and any number of tokens can be transported over a single message that simply encodes their number.

1.2 Our Contribution

Our main contribution regarding the TF problem is cast in Thm. 1 (for the formal statement, refer to the analysis sections provided in the full version [22]).

▶ **Theorem 1** (slightly informal). *For every constant $\epsilon > 0$, there exists a randomized algorithm that solves any n-node TF instance whp if at least ϵn nodes remain non-faulty indefinitely.*[5] *The algorithm sends $O(\sqrt{n \log n})$ messages per token in expectation and $O(\sqrt{n \log n} \cdot \log n)$ messages per token whp. Moreover, if the system contains at least σ tokens, then the algorithm is guaranteed to form a team in $O(\sigma + \log n)$ (asynchronous) time whp.*

We emphasize that the whp guarantees promised in Thm. 1 apply for any TF instance, regardless of the total number of tokens injected into the system that may be arbitrarily large (and not necessarily bounded as a function of n); in particular, the correctness probability remains (arbitrarily) close to 1 and does not deteriorate as the number of tokens increases. Moreover, the bounds on the number of messages per token hold already for "the first tokens", that is, even if the adversary decides to inject a few tokens in total, regardless of σ. It is also interesting to point out that our TF algorithm withstands a $(1 - \epsilon)$-fraction of faulty nodes for an arbitrarily small constant $\epsilon > 0$;[6] this is in contrast to many other problems (in message-passing with initial failures), including leader election and consensus, where a majority of faulty nodes leads to impossibility results. On the negative side, we establish the following theorem (refer to the full version [22] for the formal statement).

▶ **Theorem 2** (slightly informal). *For $2 \leq \sigma \leq n/2$, consider the simplified (one-shot) version of the TF problem, where the schedule is synchronous and it is guaranteed that exactly σ tokens are injected into the system, all at the beginning of the execution. Any algorithm that solves this problem whp must send a total of $\Omega(\max\{\sqrt{n \log n}, \sqrt{n\sigma}\})$ messages in expectation. This holds even if the messages are of unbounded size.*

1.3 Extra Features

Beyond the safety and liveness conditions presented in Sec. 1.1, it may be advantageous for TF algorithms to satisfy additional features. In this section, we introduce three such desirable features that turn out to be very useful (see Sec. 1.4), all three of them are readily satisfied by the TF algorithm promised in Thm. 1. Refer to Sec. C for further discussions of the qualities of these features.

The Forgetful Feature. In the scope of this feature, we distinguish between two types of coin tosses of a node $v \in V$: (1) the finitely many "factory coin tosses" generated by v upon the first activation event of v, before any other action of v and hence, could have been

[4] Conditioned on the assumption that $\sigma \leq n$, the nodes themselves should hold (all together) less than n^2 tokens since a node that holds at least σ tokens can perform team formation operations. Regarding the tokens in transit, the algorithm developed in the current paper is designed so that there are less than σ tokens in transit over any edge at any given time, so the total number of tokens in transit is $O(n^3)$.

[5] An event A occurs *with high probability (whp)* if $\mathbb{P}(A) \geq 1 - n^{-c}$ for any (arbitrarily large) constant c.

[6] Refer to Sec. 3.1 for a precise definition of the failure model adopted in the current paper, including the assumptions made on the faulty nodes.

generated when v was "manufactured"; (2) all other coin tosses of v, generated along the execution. In contrast to the latter, the "factory coin tosses" are considered to be hardwired into v's memory and, in particular, constitute part of v's initial state.

We say that a randomized algorithm satisfies the *forgetful feature* if it is guaranteed that every node resides in its initial state at any quiescent time (formally defined in Sec. 3.1). Thus, at quiescent times, the nodes may not record any information about their past events and actions. We view the notion of forgetful algorithms as a natural extension of the important notion of *memoryless* algorithms [57, 59] from deterministic to randomized algorithms and believe that it should be studied further, regardless of the TF problem.

The Trace-Tree Feature. A token τ injected into node $v \in V$ follows a tour in the communication graph from v to the node r at which τ is deleted (if at all) as part of a team formation event F. Somewhat informally, this tour forms a simple path π in the temporal graph that reflects the communication graph along the execution's time axis (a node $v' \in V$ may appear in π several times, but at different times). The collection of the paths π of all the tokens τ that participate in F form a temporal tree T rooted at r. A TF algorithm satisfies the *trace-tree feature* if the algorithm maintains a distributed data structure that supports broadcast and echo processes over T (it is the responsibility of "the user" to delete this data structure once it is no longer used). Since the basic description of our TF algorithm, as presented in Sec. 4, does not address this trace-tree feature, we provide the implementation details in [22].

The Accumulation Feature. Consider a TF instance and assume that the total number ℓ of tokens injected into the system satisfies $\ell \bmod \sigma = k > 0$. A TF algorithm satisfies the *accumulation feature* if it is guaranteed that the k tokens that remain in the system forever are eventually held by the same node.

1.4 Applications

We present several interesting problems that are reducible to TF. These include the classic leader election problem, for which we improve the state-of-the-art, as well as various problems that received less attention from the community so far (if at all). For each problem, we provide a sketchy description of the reduction to TF; further technical details as well as precise statements of the results obtained through these reductions are deferred to Sec. A.

Leader Election (LE). This is the fundamental one-shot problem of designating a single non-faulty node of a communication network as a leader [49, 7, 52]. We improve upon the previous results for both (1) the *implicit* version, where each node is required to know whether it is the elected leader; and (2) the *explicit* version, where, in addition to the above requirement, each non-leader knows which of its (internal) ports leads to the leader. Moreover, our algorithms are fault-tolerant (see Sec. 3.1), while the improved upon prior art [43, 44] assumes fault freedom.

Intuitively, the reduction works as follows: A logarithmic number of nodes are chosen probabilistically to serve as candidates, injecting a token into each one of them. The team size parameter is adjusted so that the number of injected tokens suffices for the formation of exactly one team whp. The node at which the team is formed is elected. In the explicit LE version, on top of the above, the elected leader notifies all other nodes.

We assume that there are at most $n(1/2 - \epsilon)$ faulty nodes for any constant $\epsilon > 0$. The run-time of our LE algorithms is $O(\log n)$ whp. Our *explicit* LE algorithm sends $O(n)$ messages in expectation and whp. This improves upon the time complexity of the best

previous algorithms with $O(n)$ messages for asynchronous explicit LE [43, 44]; their time complexity was $(\log^2 n)$ even though they assume fault-freedom. Our *implicit* LE algorithm sends $O(\sqrt{n \log n})$ messages in expectation and $O(\sqrt{n \log n} \cdot \log n)$ messages whp. This is the first algorithm with time complexity $o(n)$ for asynchronous LE (even for fault-free algorithms).

Vector Team Formation. Consider a (long-lived) generalization of the TF problem, where each token comes with a *color* from a certain color palette and a team should include a pre-specified number of tokens from each color. This problem, called *vector team formation (vTF)*, is formulated as follows: Instead of a scalar team size parameter $\sigma \in \mathbb{Z}_{>0}$, the vTF problem is defined over a *team size vector* $\vec{\sigma} \in \mathbb{Z}_{>0}^m$ whose dimension m corresponds to the size of the color palette; a team now consists of (exactly) $\vec{\sigma}(i)$ tokens of color i for each $i \in [m]$. As in the TF problem, the safety condition states that tokens can be deleted only as part of a team, whereas the liveness condition states that if the system contains at least $\vec{\sigma}(i)$ tokens of color i for each $i \in [m]$, then a team must be formed in a finite time. (The aforementioned Dungeons & Dragons example is a special case of vTF.)

To solve vTF, we shall invoke a separate copy of a TF algorithm for each color $i \in [m]$, setting $\sigma = \vec{\sigma}(i)$, and generate a "super-token" of color i whenever the i-th copy forms a team. The super-tokens can then be collected into a vTF team, exploiting the fact that the symmetry is now broken due to the different colors.

Agreement with Failures. The problem of (explicit) agreement with initial failures was defined in the seminal paper of [24] as a contrast to the setting of asynchronous crash failures proved to be impossible. A weaker version of the problem, known as *implicit* agreement, is defined in [8]. Informally, the latter problem requires that (1) at least one node must decide; (2) the decided value must be the same for all deciding nodes; and (3) the decided value must be the input of some node. This version of the agreement problem was addressed so far only in *synchronous* networks, considering both fault-free environments [8] and crash faults [40]. As implicit leader election is directly reducible to the aforementioned implicit leader election problem, we obtain an *asynchronous* implicit leader election algorithm that withstands $n(1/2 - \epsilon)$ faults in the initial failures model.

Online Gaming Platforms. This long-lived problem addresses an online gaming platform (see, e.g., [23]) in a "one-versus-one" game mode (two players compete against each other directly like in chess), where players arrive over time and should be matched with opponents. The reduction to TF works as follows: Each player is represented by a single token, which is injected into a relevant node, e.g., a nearby server, and the TF algorithm is invoked with $\sigma = 2$. When a team is formed, the trace-tree is used to establish a connection between the servers of the matched players.

Distributed Matching with Failures. In this one-shot problem, the nodes are equipped with unique IDs and an (even size) subset of the (non-faulty) nodes is marked as part of the input. The goal is to form a perfect matching in the subgraph induced by the marked nodes so that each node knows the ID of the node it is matched to. To reduce this problem to TF, intuitively, each marked node generates a token and the TF algorithm is invoked with $\sigma = 2$; the trace-tree is employed to allow the matched nodes to exchange IDs.

Robot Team Gathering. The one-shot *gathering* problem requires robots, initially positioned arbitrarily, to meet at a single point within a finite time, where the meeting point is not fixed initially [5, 48, 51, 25, 58]. We propose and solve a new problem called *robot team gathering* which is a generalization of the gathering problem. Robots are required to form teams of a given (known) size, and each team must meet at a single point within a finite time, where the meeting points are not fixed initially. In the reduction of robot team gathering (in a complete graph) to TF, each robot is represented as a token, and the TF algorithm is used to form teams.[7]

Distributed Trigger Counting. Consider a network of interconnected devices where the devices count triggers from an external source. In the one-shot *distributed trigger counting (DTC)* problem, the algorithm is required to raise an alarm when the total number of triggers counted by all the devices reaches a predefined threshold [17, 20, 37]. Intuitively, to solve DTC, each trigger generates a token, and the TF algorithm is invoked after setting the team size parameter to the threshold value.

Team Forming with Associated Information. The application is for the case where a token τ arrives with some associated information $\mathrm{AS}(\tau)$. It is required that $\mathrm{AS}(\tau)$ be available at the node where the team, that includes τ, is formed at the time of the formation. For example, $\mathrm{AS}(\tau)$ may be a piece of a secret, and σ pieces are needed and are enough to recover the secret. We further assume that the information associated with any two tokens that exist in the network simultaneously is different. This means that the size of $\mathrm{AS}(\tau)$ may be large, so only a constant number of such pieces may fit in one message (otherwise, the implementation may be straightforward even if the message size is $O(\log n)$). The reduction to (the standard version of) TF is slightly more technical, see [22].

1.5 Paper's Outline

In Sec. 2, we present the main technical challenges that arise in the study of the TF problem. Sec. 3 introduces the model, initial failures generalization, complexity measures, and basic definitions. Sec. 4 provides a high-level description of the algorithm, structured into two layers and the interface between them. Sec. 5 presents the main analysis of the algorithm, establishing safety and liveness and analyzing the message load. Sec. A contains additional technical details (omitted from Sec. 1.4) for LE as an application of our TF algorithm. Further related work is surveyed in Sec. B. Sec. C offers, a discussion on the usefulness of the TF problem, the results presented, and future directions.

Due to space considerations, some details are omitted from this extended abstract and deferred to the full version [22]. This includes technical details of the algorithm's lower layer, an analysis of the algorithm's reaction time, our lower bound (stated in Thm. 2), further details on the trace-tree mechanism mentioned in Sec. 1.4, additional applications of the TF problem beyond LE (see Sec. 1.4), and various figures, tables, and pseudocodes.

[7] It is not difficult to have the robots (each playing a token) to simulate the algorithms of the nodes, using whiteboards [19, 26] at the nodes.

2 Technical Challenges

Before we move on to presenting our solution for the TF problem, the reader may wonder about the following simple scheme: Elect a leader v (once) and following that, transport all injected tokens to v so that v handles all team formation operations. We argue that this scheme suffers from various severe shortcomings: (1) under our failure model (formally presented in Sec. 3.1), the leader v may become faulty at any quiescent time, causing all subsequently injected tokens to get lost; (2) since our model does not include "spontaneous wake-ups", it is not clear which nodes participate in the leader election stage; (3) leader election is impossible if we allow a $(1-\epsilon)$-fraction of the nodes to be faulty; and (4) the scheme does not satisfy the forgetful feature (as defined in Sec. 1.3).

The aforementioned shortcomings highlight some of the technical challenges we had to overcome when devising our TF algorithm. To discuss those, assume for now that the team size parameter σ is a large constant and consider a token τ injected into some node v. The first task faced by v is to look for other (nodes that hold) tokens – if there are at least $\sigma - 1$ of those, then a team should be formed. Doing so in a communication efficient manner is a challenge, especially under an asynchronous scheduler when a large fraction of the nodes may be faulty (and thus, never respond to incoming messages). We tackle this challenge by constructing a "probabilistic quorum system" with quorums of size $O(\sqrt{n \log n})$ (using the "factory coin tosses"), thus ensuring that any two token holding nodes have at least one non-faulty node in the intersection of their quorums.

Once node v has identified other token holding nodes, the next task is to determine who transports the tokens to whom, raising a symmetry breaking challenge: a naive approach may cause v to transport its token to v' while v' transports its own token to v'' and so on. To prevent long token transportation chains, we adopt a "star-shaped" transportation pattern: the token holding nodes run consecutive phases and in each phase, assume a center or an arm role at random; each arm node v_a then tries to transport its token to a center node v_c that is "ready to accept" v_a's token. For this idea to succeed, one has to lower-bound the probability for an arm node to identify an appropriate center node in each phase. However, as we cannot synchronize between the phases of different nodes, ensuring such a probability lower bound turns out to be eminently challenging. We resolve this challenge by incorporating a "selective waiting mechanism" inspired by a technique introduced in [9] (for the entirely different task of constructing a maximal independent set in a general graph); refer to Sec. 4 for details.

3 Preliminaries

3.1 The Computational Model

Fix some randomized TF algorithm `Alg`. Recall our assumption that the adversary assigns a finite delay to each message of `Alg`. The only constraint we impose on the adversary in this regard is that the messages sent from a node $v \in V$ to a node $v' \in V$ are delivered in FIFO order. For the sake of the runtime analysis (more on that later), we scale the time axis so that the message delays are up-bounded by 1 time unit; we emphasize that the nodes themselves have no notion of time.

Event Driven Executions. The execution of `Alg` advances through the continuous time axis $\mathbb{R}_{>0}$ in an event driven fashion so that a node $v \in V$ is activated at time $t > 0$ if (and only if) one of the following two *activation events* occurs at time t: (1) a message is delivered to v; or (2) one or more tokens are injected into v. When activated, the actions of v under

`Alg` include (I) reading the new incoming message (if any); (II) local computation that may involve (private) random coin tosses; (III) any number of team formation operations; and (IV) sending messages to (a subset of) v's neighbors.[8] It is assumed that actions (I)–(IV) are performed atomically (i.e., they take zero time). We emphasize that the nodes never act unless activation events (1) or (2) occur (in particular, in the scope of the TF problem, there are no "spontaneous wake-ups").

Assume without loss of generality that the activation events are isolated so that no two of them occur at the same time; the times at which these activation events occur are referred to as *activation times*. Time $t > 0$ is regarded as *quiescent* if (1) t is not an activation time; (2) there are no messages in transit at time t; and (3) the system contains 0 tokens at time t.

For a time $t > 0$, let $t- = t - dt$ and $t+ = t + dt$ be the times immediately before and immediately after t, respectively, where dt is chosen so that the time interval $[t-, t+]$ contains at most one activation event. Given an object `obj`, let $\texttt{obj}^t$ denote the state of `obj` at time t, adhering to the convention that $\texttt{obj}^t = \texttt{obj}^{t-}$ for all $t > 0$;[9] in particular, if t is an activation time and the state of `obj` changes at time t, then $\texttt{obj}^t$ is taken to be the state before the change. Notice that our convention implies that instantaneous changes to `obj`, that are done as part of the local computation and do not persist beyond time t, are not reflected in $\texttt{obj}^t$.

Fault Tolerance and Adversarial Policies. We adopt a generalization of the *initial failures* model [18, 24, 60]. Under the initial failures model, the adversary selects a node subset $F \subset V$ at time 0 (i.e., right before the execution commences) and turns the status of all the nodes in F to *faulty*, thus preventing them from participating in the execution. In particular, faulty nodes do not send messages and in the context of the TF problem, no tokens are injected into faulty nodes. A message μ delivered to a faulty node is lost (i.e., it is ignored and does not trigger an activation event). If tokens are transported over a lost message μ, then these tokens are assumed to remain in "limbo" forever, which may prevent the algorithm from satisfying the liveness condition; it is the responsibility of the algorithm designer to avoid such scenarios.

In the (new) generalized version of the initial failures model, considered in the current paper, the adversary may *toggle* the status of a node $v \in F$ from faulty to non-faulty and vice versa, so that v may participate in the execution in certain time intervals and not participate in others. The decisions of the adversary in this regard are subject to the following constraint: the faulty/non-faulty status of v may be toggled only at quiescent times. For a precise description of the failure model, if v becomes faulty at time $t \geq 0$ and remains faulty until time $t' > t$, at which it becomes non-faulty, then the memory image of v at time t' is assumed to be identical to that of time t. To avoid confusion, we subsequently refer to the nodes in F as *fragile* (regardless of their faulty/non-faulty status at a specific time of the execution), emphasizing that the non-fragile nodes remain non-faulty throughout the execution.

Notice that due to the asynchronous nature of the system, it may be impossible for a non-faulty node to distinguish its faulty neighbors from the "slow responding" ones. Moreover, the fragile nodes do not know that they are fragile and when a fragile node becomes non-faulty, it runs the same algorithm as the non-fragile nodes. The generalization of the classic initial failures model is different from the crash failures model, where a node may fail at any time. As pointed out in [24], considering initial failures is known to open the gate for asynchronous algorithms that are impossible under crash failures.

---

[8] We allow v to send multiple messages to the same neighbor (in practice, those can be piggybacked).
[9] Objects in this regard include local variables as well as global objects defined for the analysis.

With this failure model, the adversary has the following roles: (1) At time 0, the adversary determines the subset $F \subset V$ of fragile nodes and (without loss of generality) turns their status to faulty. (2) For time $t = 0$ and for every activation time $t > 0$, at time $t+$, the adversary determines the next activation event (if any) including its type (message delivery or token injection), location, and time $t' > t$. (3) For every quiescent time $t > 0$ and for every node $v \in F$, the adversary determines (at time t) if the faulty/non-faulty status of v is toggled at time t (notice that this is *not* an activation event). We emphasize that tokens can be injected into node v at time t only if v is non-faulty at time t; tokens are never injected into faulty nodes.

The adversary is *adaptive*, namely, its decisions at time t may be based on the nodes' actions before time t (however, without knowing the nodes' future coin tosses). Consequently, the execution of `Alg` can be represented as an extensive form game with incomplete information [30], where the adversary is the only strategic player; that is, a rooted tree whose (internal) vertices are partitioned into a set of adversarial moves, in which the adversary makes its decisions, and a set of chance moves, each corresponding to the coin tosses of an activated node in some activation event. Terminology-wise, a mechanism that makes the decision in each adversarial move of the aforementioned extensive form game is referred to as an *adversarial policy*. Notice that by fixing an adversarial policy, `Alg`'s execution becomes a random variable, fully determined by the nodes' coin tosses.

Performance Measures. The quality of `Alg` is evaluated by means of two performance measures. First and foremost, we wish to bound the number of messages that `Alg` sends per token in expectation and whp. Care is needed in this regard since the adaptive nature of the adversary implies that the number of tokens injected into the system may depend, by itself, on the nodes' coin tosses. Thus, the number of messages per token is, in general, the ratio of two random variables that may exhibit complex dependencies, making it difficult to reason about. To resolve this difficulty, we formulate our main performance measure through the notion of *message load*, defined as follows.

For $\ell \in \mathbb{Z}_{>0}$, let $\mathcal{A}_\ell$ be the family of adversarial policies that inject at most ℓ tokens into the system (throughout the execution). We say that `Alg`'s *message load* is M in expectation (resp., whp) if M is the smallest real such that for every $\ell \in \mathbb{Z}_{>0}$ and for every adversarial policy in $\mathcal{A}_\ell$, it is guaranteed that `Alg` sends at most $M \cdot \ell$ messages in expectation (resp., whp).[10]

We are also interested in the time it takes for `Alg` to form a team once sufficiently many tokens are present. To this end, we say that `Alg`'s *reaction time* is R if R is the smallest real such that for every time $t > 0$, if the system contains at least σ tokens at time t, then a team is formed by time $t + R$ whp.

3.2 The Primary-Utility Graph

The algorithms developed in this paper are designed so that each node $v \in V$ simulates a virtual *primary node* $p(v)$ and a virtual *utility node* $u(v)$; let $P = \{p(v) : v \in V\}$ and $U = \{u(v) : v \in V\}$ denote the sets of primary and utility nodes, respectively. Although the primary node $p(v)$ and utility node $u(v)$ are simulated by the same "physical" node $v \in V$,

[10] We note that the message load provides a bound on the number of messages sent (per token) in "finite prefixes" of the execution under *any* adversarial policy, including those that do not belong to $\bigcup_{\ell>0} \mathcal{A}_\ell$.

it is convenient to address them as standalone computational entities that participate in the execution by receiving and sending messages independently of v. To this end, tokens injected into v are regarded as injected into the primary node $p(v)$.

In terms of the failure model, the primary node $p(v)$ and utility node $u(v)$ are regarded as *fragile* if the corresponding "physical" node $v \in V$ is fragile. Let $F_P = \{p(v) \in P : v \in F\}$ and $F_U = \{u(v) \in U : v \in F\}$ denote the sets of fragile primary and utility nodes, respectively. The faulty/non-faulty status of the fragile primary node $p(v) \in F_P$ and fragile utility node $u(v) \in F_U$ is assumed to be inherited from that of the corresponding fragile "physical" node $v \in F$.[11]

A key design feature of our algorithms is that the entire message exchange is confined to an overlay bipartite graph $G_{PU} = (P, U, E_{PU})$, referred to as the *primary-utility graph*. The edge set E_{PU} of the primary-utility graph is constructed randomly by selecting each node pair $(p, u) \in P \times U$ to be included in E_{PU} independently with probability $c \cdot \sqrt{\frac{\log n}{n}}$, where $c > 0$ is a constant to be derived from the analysis;[12] this random selection is made by each primary node p upon its first activation event and remains fixed throughout the execution. Notation-wise, for a primary node $p \in P$ and a utility node $u \in U$, let $U(p) = \{u' \in U : (p, u') \in E_{PU}\}$ and $P(u) = \{p' \in P : (p', u) \in E_{PU}\}$. The key probabilistic properties of the primary-utility graph are cast in the following lemma (proof deferred to the full version [22]); in the remainder of this paper, we condition on the event that these properties are satisfied.

▶ **Lemma 1.** *The primary-utility graph $G_{PU} = (P, U, E_{PU})$ satisfies the following two properties whp:*
(1) $(U(p) \cap U(p')) - F_U \neq \emptyset$ *for every $p, p' \in P$; and*
(2) $|U(p)|, |P(u)| \leq O(\sqrt{n \log n})$ *for every $p \in P$ and $u \in U$.*

4 The Algorithm

In this section, we present our TF algorithm, referred to as $\texttt{Alg}_{\text{TF}}$. The algorithm attempts to (locally) gather tokens, that may be distributed over multiple primary nodes, into a single primary node and perform team formation(s) if the number of gathered tokens is large enough.

Under $\texttt{Alg}_{\text{TF}}$, each primary node $p \in P$ maintains the local variable $p.\texttt{tok} \in \mathbb{Z}_{\geq 0}$ that counts the number of tokens held by p. Recalling that $p.\texttt{tok}^t$ is the value of $p.\texttt{tok}$ at time $t > 0$, let $\mathcal{B}^t = \{p \in P : p.\texttt{tok}^t > 0\}$, referring to the nodes in $\mathcal{B}^t$ as *busy*. The algorithm is designed so that a utility node $u \in U$ may hold tokens only instantaneously, when the tokens are transported, through u, between two primary nodes in $P(u)$.

Fix some $p \in P$ and $t > 0$ and suppose that $p \in \mathcal{B}^t$ and that t is an activation time of p. If $p.\texttt{tok}^t < \sigma$, then p may decide, at time t, to transport the tokens it holds by sending a message μ, that carries these tokens, to another primary node $p' \in P$, through a utility node $u \in U(p) \cap U(p')$; the algorithm is designed so that this may happen only if p' is busy at time t and remains busy (at least) until message μ is delivered to p'. If $p.\texttt{tok}^t \geq \sigma$, then p may decide, at time t, to eliminate some of the tokens it holds, in which case, p performs $\lfloor p.\texttt{tok}^t/\sigma \rfloor$ team formations (i.e., as many team formations as possible). We emphasize that p will never transport tokens as long as $p.\texttt{tok} \geq \sigma$ and that if p does transport the $(p.\texttt{tok} < \sigma)$ tokens it holds, then all $p.\texttt{tok}$ tokens are transported together and p becomes non-busy.

[11] This assumption is made only for the sake of simplicity; as far as our algorithms and analyses go, the faulty/non-faulty status of $p(v)$ and $u(v)$ can be decoupled.
[12] With a slight abuse of notation, we often use p (resp., u) as a placeholder for a general primary (resp., utility) node.

If p performs team formation(s) at time t and $p.\texttt{tok}^t \bmod \sigma = k > 0$, then p is left with k "remainder tokens". To simplify the presentation, we treat these k "remainder tokens" as if they are injected into p, as fresh tokens, soon after time t (and strictly before the next "original" activation event); refer to the injections of such "remainder tokens" as *fake injections*. In particular, we assume that if p performs team formation(s) at time t, then $p.\texttt{tok}^{t+} = 0$, hence $p \notin \mathcal{B}^{t+}$. Put differently, as long as $p \in \mathcal{B}$, the variable $p.\texttt{tok}$ is a non-decreasing function of t.[13]

Two-Layer Structure. To simplify the algorithm's presentation, we divide it, logically, into two layers: a lower *channel layer* and an upper *principal layer*. Rather than sending messages directly, the principal layer in (busy) primary nodes uses the service of the channel layer for communication. This service, called *channels*, bears similarities to virtual circuits such as TCP connections. As in the case of TCP connections, a channel χ between two primary nodes $p, p' \in P$ can be released, and sometimes later, a new channel χ' between p and p' may be created. An important feature of the channels is that a message sent from p to p' as part of χ does not arrive as part of χ'; this is formalized, together with several other important assurances of the channel layer, in Grnt. 1–6 below.

A reader who is familiar with the nuts and bolts of virtual circuits knows that the task of ensuring such properties is messy, sometimes non-trivial, but nonetheless possible. This is the reason we defer the description of the implementation of the channel layer to the full version [22], whereas the principal layer's implementation is presented in the current section. For now, let us just say that in the channel layer, each primary node $p \in P$ updates all the utility nodes in $U(p)$ whenever its status changes from busy to non-busy or vice versa. A (non-faulty) utility node $u \in U$ chooses two primary nodes in $P(u)$ that reported they are busy, and creates a channel between them. This channel is released by u when (and only when) u hears that one of the channel's primary nodes is no longer busy. We now provide a more formal description of the channels and the interface between the two layers.

The Channels. The role of the channels is to enable (duplex) communication among (unordered) pairs of busy primary nodes. For $p, p' \in P$, each $\{p, p'\}$-channel χ is associated with a utility node $u \in U(p) \cap U(p')$, referred to as the channel's *mediator* that relays the messages that p and p' exchange with each other as part of χ, referred to hereafter as *relayed messages*. The algorithm is designed so that for every $u \in U$ and $p, p' \in P$, at any given time, the system includes at most one channel mediated by u and zero or more $\{p, p'\}$-channels, each mediated by its own utility node in $U(p) \cap U(p')$.

To keep track of the channels, a primary node $p \in P$ maintains the local variable $p.\texttt{meds} \subseteq U(p)$ that stores the mediators of the $\{p, \cdot\}$-channels; a utility node $u \in U$ maintains the local variable $u.\texttt{chan} \subseteq P(u)$ defined so that $u.\texttt{chan} = \{p, p'\}$ if u mediates a $\{p, p'\}$-channel, and $u.\texttt{chan} = \emptyset$ otherwise. For a $\{p, p'\}$-channel χ mediated by u, we refer to the time $t > 0$ at which u sets $u.\texttt{chan} \leftarrow \{p, p'\}$ as the *creation time* of χ; we refer to the earliest time $\bar{t} > t$ such that $u.\texttt{chan}^{\bar{t}+} \neq \{p, p'\}$ as the *release time* of χ.

We emphasize that channel χ is created (resp., released) once and if the mediator u creates (resp., releases) a $\{p, p'\}$-channel χ' at time $t' \neq t$ (resp., $\bar{t}' \neq \bar{t}$), then χ' and χ are considered to be two different channels. Notice that the placeholders χ and χ' are introduced for the sake of the discussion and we do not assume that the primary nodes p and p' agree on "common names" for the $\{p, p'\}$-channels.

[13] Throughout, we omit the superscript t from $\texttt{obj}^t$ when we wish to address the dynamic nature of the object $\texttt{obj}$ whose state may vary over time.

Consider some primary node $p \in P$ and utility node $u \in U(p)$. A key feature of the algorithm is that if $I \subset \mathbb{R}_{>0}$ is a maximal time interval such that $u \in p.\texttt{meds}^t$ for all $t \in I$, then all relayed messages that p sends to or receives from u during I belong to the same $\{p, p'\}$-channel χ (mediated by u) for some primary node $p' \in P(u) - \{p\}$; we refer to χ and p' as the *u-channel* and *u-peer*, respectively, of p during I. Notation-wise, let $\text{channel}^t(p, u)$ and $\text{peer}^t(p, u)$ be the operators that return the u-channel and u-peer, respectively, of p at time t if $u \in p.\texttt{meds}^t$, and $\perp$ otherwise.

The algorithm is designed so that if χ is a $\{p, p'\}$-channel mediated by u, then the set $\{t \in \mathbb{R}_{>0} : \text{channel}^t(p, u) = \chi\}$ is either empty or forms a single interval of the time axis. This is in contrast to the set $\{t \in \mathbb{R}_{>0} : \text{peer}^t(p, u) = p'\}$ that may form multiple intervals, each corresponding to a different $\{p, p'\}$-channel mediated by u.

Notice that $\text{channel}^t(p, u) = \chi \neq \perp$ does not imply that channel χ still exists at time t as χ may have been released by u at time $t - 1 \leq t' < t$ (which will be observed by p at time $t' < t'' \leq t' + 1$). Moreover, while one may hope that the formula $\text{peer}^t(p, u) = p' \iff \text{peer}^t(p', u) = p$ is satisfied "most of the time", it cannot be satisfied all the time; indeed, given a $\{p, p'\}$-channel χ, since the variables $p.\texttt{meds}$ and $p'.\texttt{meds}$ are updated asynchronously, we cannot expect the aforementioned formula to be satisfied "shortly after" (resp., "shortly before") χ is created (resp., released). Grnt. 4 ensures that these inconsistencies do not introduce "misunderstandings".

The Interface between the Layers. The channel layer is responsible for maintaining the channels by updating the `meds` and `chan` variables and for handling the delivery of relayed messages among peers. The principal layer governs the token gathering process among the busy primary nodes that communicate with their peers over the channels. For a primary node $p \in P$, the main component in the interface between the two layers is the set $\mathcal{C}^t(p) = \{\text{channel}^t(p, u) : u \in p.\texttt{meds}^t\}$ that captures p's channels at time $t > 0$. It is assumed that node p's channel layer maintains the set $\mathcal{C}(p)$ while hiding its implementation details; node p's principal layer then uses the channels in $\mathcal{C}(p)$ to exchange relayed messages with the principal layer of p's peers.

A primary node $p \in P$ is regarded as *operational* at time $t > 0$ if $\mathcal{C}^t(p) \neq \emptyset$; let $\mathcal{OP}^t$ be the set of operational primary nodes at time t. A $\{p, p'\}$-channel χ is regarded as *operational* at time $t > 0$ if $\chi \in \mathcal{C}^t(p) \cap \mathcal{C}^t(p')$; let $\mathcal{OC}^t$ be the set of operational channels at time t. These notions facilitate the formulation of the key assurances that the channel layer provides to the principal layer, stated in Grnt. 1–6 (established in the full version [22]).

▶ **Guarantee 1.** *$\mathcal{OP}^t \subseteq \mathcal{B}^t$ for every $t > 0$.*

▶ **Guarantee 2.** *For every $t_0 > 0$, if $|\{p \in P : p \in \mathcal{B}^t \text{ for all } t_0 < t \leq t_0 + 4\}| \geq 2$, then there exists $t_0 < t \leq t_0 + 4$ such that $\mathcal{OC}^{t+} \neq \emptyset$.*

▶ **Guarantee 3.** *Consider a $\{p, p'\}$-channel χ and time $t > 0$, and assume that $\chi \in \mathcal{OC}^t$. Then, $\chi \in \mathcal{OC}^{t+}$ if and only if $p, p' \in \mathcal{B}^{t+}$. Moreover, if $p \notin \mathcal{B}^{t+}$, then $\chi \notin \mathcal{C}^{(t+2)+}(p')$.*

▶ **Guarantee 4.** *The following conditions are satisfied for every primary node $p \in P$, time $t > 0$, and $\{p, p'\}$-channel $\chi \in \mathcal{C}^t(p)$: (I) If the principal layer of p receives a relayed message μ over χ at time t, then μ was sent over χ by the principal layer of p' at time $t - 2 \leq t' < t$ (in other words, μ could not have originated from some past/future channel of p with the same mediator). (II) If the principal layer of p sends a relayed message μ over χ at time t, then either (II.a) μ is received over χ by the principal layer of p' at time $t < t' \leq t + 2$; or (II.b) μ becomes irrelevant because p' turned from busy to non-busy, leading to the removal of χ from $\mathcal{C}(p')$.*

▶ **Guarantee 5.** $|\mathcal{C}^t(p)| \leq O(\sqrt{n \log n})$ *for every* $p \in P$ *and* $t > 0$.

▶ **Guarantee 6.** *The channel layer sends* $O(\sqrt{n \log n})$ *messages per token.*

The Principal Layer

We now turn to describe the implementation of the principal layer, building on the interface assured by Grnt. 1–6 (pseudocode is provided in the full version [22]). Recall that this layer is implemented over the (busy) primary nodes $p \in P$ that exchange relayed messages with their peers over the channels in $\mathcal{C}(p)$ (maintained by the channel layer); the utility nodes do not participate in this layer and are abstracted away from its description. The policy of the principal layer for a non-operational node $p \in P$ is straightforward: p performs (as many as possible) team formation operations if $p.\texttt{tok} \geq \sigma$, and does nothing otherwise. The interesting part of the principal layer algorithm addresses the operational nodes that "compete" with their peers over the right to serve as the local gathering point for the tokens. To this end, if $p.\texttt{tok}, p'.\texttt{tok} < \sigma$ for some $p, p' \in \mathcal{OP}$ and $\mathcal{C}(p) \cap \mathcal{C}(p') \neq \emptyset$, then (local) symmetry breaking is needed to determine whether p transports its tokens to p' or p' transports its tokens to p (or neither).

The principal layer resolves this symmetry breaking challenge by following a "star-shaped" token transportation pattern. Specifically, an operational node $p \in \mathcal{OP}$ runs consecutive *phases* so that in each phase, p assumes a *phase type* that can be either *center* or *arm*. This phase type is determined by p via an (unbiased) coin toss when the phase begins and is stored in the local variable $p.\texttt{phase-type} \in \{\mathsf{center}, \mathsf{arm}\}$ that p maintains. The crux of the (dynamic) classification into phase types is that a center node attempts to collect the tokens from its arm peers so that a phase of a center node p is successful if one or more of p's arm peers transport their tokens to p; a phase of an arm node p is successful if p transports its tokens to exactly one of its center peers.

Each phase ϕ of node p proceeds according to the following *request-response mechanism*: When ϕ begins, node p sends a *request message* over each channel in $\mathcal{C}(p)$; phase ϕ ends once p has received a *response message* for each request message sent at the beginning of ϕ. (As discussed soon, ϕ may end abruptly if p decides to transport its tokens to one of its peers, in which case, p becomes non-busy and hence, also non-operational.) To implement this request-response mechanism, p maintains the local variable $p.\texttt{awaiting-resp}(\chi) \in \{\mathsf{true}, \mathsf{false}\}$ for each $\chi \in \mathcal{C}(p)$, setting $p.\texttt{awaiting-resp}(\chi) \leftarrow \mathsf{true}$ when a request message is sent over χ and resetting $p.\texttt{awaiting-resp}(\chi) \leftarrow \mathsf{false}$ when a response message is received over χ.

When phase ϕ ends, node p checks if the inequality $p.\texttt{tok} \geq \sigma$ holds and if it does, performs (as many as possible) team formation operations. To simplify the presentation, we treat any tokens injected into p during phase ϕ as if they are injected when ϕ ends; we emphasize that the injected tokens are accounted for when checking if the inequality $p.\texttt{tok} \geq \sigma$ holds.[14]

The type of the request messages depends on the phase type of their sender: the requests of the center nodes are TokensPlease messages, whereas the requests of the arm nodes are Waiting messages. The reaction of the operational nodes to an incoming request, including the type of the corresponding response message, depends on the phase type of the receiver as well as on the type of the incoming request. To this end, consider an operational node $p \in \mathcal{OP}$ that receives a request μ over channel $\chi \in \mathcal{C}(p)$ during phase ϕ.

[14] To adhere to the formal event driven model defined in Sec. 3.1, that forbids concurrent activation events, one can move the token injection event "shortly after" the end of phase ϕ so that this event triggers the team formation operations (if any) or the beginning of the subsequent phase (if p is still operational).

Assume first that $\mu = \mathsf{TokensPlease}$. If ϕ is a center phase, then p reacts by sending a $\mathsf{NoTransport}$ response. If ϕ is an arm phase, then p reacts by sending a $\mathsf{Transport}(p.\mathtt{tok})$ response that transports the tokens held by p (excluding those injected during ϕ) to the peer of p at the other end of χ. Consequently, p becomes non-busy and the channels in $\mathcal{C}(p)$ are released (the channel layer takes care of that), making p non-operational; this is the one exception to the request-response mechanism, where phase ϕ ends abruptly without waiting for all the responses (since $\mathcal{C}(p)$ is emptied, Grnt. 4 ensures that the "missing responses" are dropped and do not interfere with future channels of p).

Now, assume that $\mu = \mathsf{Waiting}$. If ϕ is an arm phase, then p reacts by sending a GoOn response. If ϕ is a center phase, then p delays its response until the current phase ϕ ends and the phase type of the next phase ϕ' of p is revealed. At this stage, if ϕ' is a center phase, then p "forgets about" μ and continues as usual with ϕ'; in particular, p sends $\mathsf{TokensPlease}$ requests over all channels in $\mathcal{C}(p)$, including χ – this $\mathsf{TokensPlease}$ request over χ plays a key role because it is guaranteed to prompt the peer p' of p on the other end of χ to transport its tokens if p' did not do so already. If ϕ' is an arm phase, then p first sends a GoOn message over χ and then, continues as usual with ϕ' (sending $\mathsf{Waiting}$ requests over all channels in $\mathcal{C}(p)$). To implement this policy, p maintains the local variable $p.\mathtt{delaying\text{-}resp}(\chi) \in \{\mathsf{true}, \mathsf{false}\}$ for each channel $\chi \in \mathcal{C}(p)$, setting $p.\mathtt{delaying\text{-}resp}(\chi) \leftarrow \mathsf{true}$ when p receives a $\mathsf{Waiting}$ message over χ as part of a center phase ϕ and resetting $p.\mathtt{delaying\text{-}resp}(\chi) \leftarrow \mathsf{false}$ at the beginning of the next phase ϕ', after a GoOn response was sent over χ if needed (i.e., if ϕ' is an arm phase).

We emphasize that the $p.\mathtt{awaiting\text{-}resp}(\chi)$ and $p.\mathtt{delaying\text{-}resp}(\chi)$ variables of p are maintained only for channels $\chi \in \mathcal{C}(p)$. In particular, if channel χ is added to (resp., removed from) $\mathcal{C}(p)$ in the midst of phase ϕ, then the variables $p.\mathtt{awaiting\text{-}resp}(\chi)$ and $p.\mathtt{delaying\text{-}resp}(\chi)$ are created (resp., deleted) with it, where both of them are initialized to false.

5 Analysis of the Algorithm – Correctness and Message Load

Throughout this section, we fix some adversarial policy $\mathtt{Adv}$ and analyze the execution of algorithm $\mathtt{Alg}_{\mathrm{TF}}$ under $\mathtt{Adv}$. We start with some useful definitions, followed by Obs. 2 and 3, that capture basic features of $\mathtt{Alg}_{\mathrm{TF}}$, and Lem. 4, that serves as the cornerstone of the entire analysis. Sec. 5.1 is then dedicated to establishing the algorithm's correctness while the message load analysis is presented in Sec. 5.2. The reaction time analysis is deferred to as it builds on a certain implementation feature of the channel layer, presented in Throughout the analysis, we condition on the event that the assertion of Lem. 1 holds. Proofs are omitted from this extended abstract and appear in the full version [22].

We say that a primary node $p \in P$ *retires* at time $t > 0$ if $p \in \mathcal{OP}^t - \mathcal{B}^{t+}$, observing that this holds if and only if a phase ϕ of p ends at time t and either (1) ϕ is an arm phase that ends abruptly and p sends a $\mathsf{Transport}$ message at time t; or (2) p performs team formation(s) at time t. We say that a $\{p_1, p_2\}$-channel χ *retires* at time $t > 0$ if $\chi \in \mathcal{OC}^t - \mathcal{OC}^{t+}$, observing that by Grnt. 3, this holds if and only if $\chi \in \mathcal{OC}^t$ and p_i retires at time t for some (exactly one) $i \in \{1, 2\}$. Notice that if a channel χ becomes operational (resp., retires) at time $t > 0$ (resp., at time $t' > t$), then χ is created (resp., released) by its mediator strictly before time t (resp., strictly after time t').

For $p \in P$ and $t > 0$, let $\Phi_p(t)$ be the operator that returns the earliest time $t' > t$ such that a phase of p ends at time t' if $p \in \mathcal{OP}^{t+}$, and t otherwise (Obs. 2 ensures that this operator is well defined). Notice that if we fix time $t > 0$, including all coin tosses up to

(excluding) time t, then $\Phi_p(t)$ is a random variable that depends on the coin tosses from time t onward. Notice further that while the condition $p \in \mathcal{OP}^t \wedge \Phi_p(t) = t$ does not imply that p retires at time t (it may be the case that p is no longer operational at time $t+$ although it is still busy), this condition does imply that every channel $\chi \in \mathcal{C}^t(p)$ retires at or before time t.

The $\Phi_p(\cdot)$ operator is extended inductively as follows: let $\Phi_p^0(t) = t$; for $i > 0$, let $\Phi_p^i(t) = \Phi_p(\Phi_p^{i-1}(t))$. This extension gives us a convenient handle for reasoning about events that occur "within the next i phases" of p.

▶ **Observation 2.** *A phase that begins at time $t > 0$ is guaranteed to end at or before time $t + 8$.*

▶ **Observation 3.** *If a* Transport *response is in transit from a primary node p to a primary node p' over a $\{p, p'\}$-channel χ at time $t > 0$, then $p' \in \mathcal{B}^t$ and $\chi \in \mathcal{C}^t(p')$.*

▶ **Lemma 4.** *Fix time $t_0 > 0$ and consider a $\{p_1, p_2\}$-channel $\chi \in \mathcal{OC}^{t_0}$. With probability at least $1/16$, channel χ retires during the time interval $[t_0, \min_{i\in\{1,2\}} \Phi_{p_i}^3(t_0)]$.*

5.1 Safety and Liveness

In this section, we establish the correctness of $\texttt{Alg}_{\text{TF}}$, proving that it satisfies the safety and liveness conditions. The former condition holds trivially as tokens are deleted only during team formation operations, so the remainder of this section is dedicated to the latter.

Consider time $t_0 > 0$ and assume that the system contains at least σ tokens at time t_0. For time $t \geq t_0$, node $p \in P$, and $\{p, p'\}$-channel $\chi \in \mathcal{C}^t(p)$, let $R^t(p, \chi)$ be the number of tokens transported over χ towards p at time t. Notice that $R^t(p, \chi) = k > 0$ if and only if there is a Transport(k) message in transit from p' to p over χ at time t. For time $t \geq t_0$ and node $p \in P$, let $R^t(p) = \sum_{\chi\in\mathcal{C}^t(p)} R^t(p, \chi)$ if $p \in \mathcal{B}^t$, and $R^t(p) = 0$ otherwise.

By Obs. 2, a primary node that holds at least σ tokens is guaranteed to form a team in $O(1)$ time. Grnt. 4 ensures that every Transport message reaches its destination in $O(1)$ time. Therefore, if $p.\texttt{tok}^t + R^t(p) \geq \sigma$, then p is guaranteed to form a team by time $t + O(1)$.

Define the potential function $\psi : \mathbb{R}_{>0} \to \mathbb{Z}_{\geq 0}$ as

$$\psi(t) = \sum_{p\in P} (p.\texttt{tok}^t + R^t(p) - 1)$$

and notice that if $\psi(t) \geq (\sigma - 1)n$, then $p.\texttt{tok}^t + R^t(p) \geq \sigma$ for some $p \in P$, hence a team is guaranteed to be formed by time $t + O(1)$. Obs. 3 ensures that if no teams are formed during the time interval $I = [t_0, \cdot] \subset \mathbb{R}_{>0}$, then the function $\psi(t)$ is non-decreasing in I. Moreover, a primary node $p \in P$ retires at time $t \geq t_0$ without forming any team if and only if p sends a Transport message at time t, implying that $\psi(t+) = \psi(t) + 1$. So, it suffices to show that as long as no teams are formed, the number of node retirements must increase.

By Obs. 2 and Lem. 4, we know that if $\mathcal{OC}^t \neq \emptyset$, then at least one primary node is certain to retire in finite time with probability 1. The liveness proof is completed by observing that if the system contains $k \geq \sigma$ tokens and there are no tokens in transit, then either (I) all k tokens are held by a single busy node that forms a team in $O(1)$ time; or (II) the k tokens are distributed over multiple busy nodes, in which case, an operational channel is certain to be generated in $O(1)$ time by Grnt. 2.

▶ **Theorem 5.** $\texttt{Alg}_{\text{TF}}$ *satisfies the safety and liveness conditions for any TF instance.*

5.2 Message Load

We now turn to analyze the message load of $\texttt{Alg}_{\text{TF}}$ in expectation and whp. To this end, fix some $\ell \in \mathbb{Z}_{>0}$ and assume that the adversarial policy $\texttt{Adv}$ injects at most ℓ tokens throughout the execution (i.e., $\texttt{Adv} \in \mathcal{A}_\ell$ in the language of Sec. 3.1). The analysis starts with bounding the number of relayed messages sent over non-operational channels.

▶ **Lemma 6.** *The total number of relayed messages sent throughout the execution over channels χ while $\chi \notin \mathcal{OC}$ is $O(\ell \cdot \sqrt{n \log n})$.*

Combined with Grnt. 6, we conclude that with the exception of relayed messages sent over operational channels, the total number of messages sent by $\texttt{Alg}_{\text{TF}}$ is $O(\ell \cdot \sqrt{n \log n})$, so it remains to bound the number of former messages. To this end, we establish Lem. 7 and Lem. 8.

▶ **Lemma 7.** $\left|\bigcup_{t>0} \mathcal{OC}^t\right| \leq O(\ell \cdot \sqrt{n \log n})$.

▶ **Lemma 8.** *Consider a $\{p, p'\}$-channel χ and let Y_χ be a random variable that counts the number of relayed messages sent over χ while $\chi \in \mathcal{OC}$. There exist constants $\alpha, \beta > 0$ such that $\mathbb{P}(Y_\chi > y) < \beta e^{-\alpha y}$ for every $y \in \mathbb{Z}_{\geq 0}$.*

For a channel χ, let Y_χ be the random variable in Lem. 8. Recalling that the expected value of an exponentially decaying random variable is bounded by $O(1)$, we can combine Lem. 7 and 8 to deduce, by the linearity of expectation, that $\texttt{Alg}_{\text{TF}}$ sends $O(\ell \cdot \sqrt{n \log n})$ messages in expectation.

At this stage, one may be tempted to advance towards a whp bound by assuming that the random variables Y_χ of different channels χ are independent. Unfortunately, this assumption is wrong: the number of relayed messages sent over different channels in $\mathcal{OC}$ may be strongly correlated. To resolve this difficulty, we use the following lemma, stating that the average of finitely many exponentially decaying random variables is also exponentially decaying even if the original random variables exhibit complex dependencies; we will not be surprised to learn that this lemma is known from the existing literature, however, we could not find it anywhere and therefore, provide a standalone proof (deferred to the full version [22]).

▶ **Lemma 9.** *Fix some $\alpha, \beta > 0$ and let $X_1, \ldots X_m$ be (not necessarily independent) random variables over $\mathbb{R}_{>0}$ such that $\mathbb{P}(X_i > x) < \beta e^{-\alpha x}$ for every $i \in [m]$ and $x > 0$. For every $\epsilon > 0$, there exists $\beta' = \beta'(\beta, \epsilon) > 0$ such that $\mathbb{P}(\bar{X} > x) < \beta' e^{-\frac{\alpha x}{1+\epsilon}}$ for every $x > 0$, where $\bar{X} = \frac{1}{m} \sum_{i \in [m]} X_i$.*

Let $m = O(\ell \cdot \sqrt{n \log n})$ be the bound promised in Lem. 7. By applying Lem. 9 to the (at most) m random variables in $\{Y_\chi\}_{\chi \in \bigcup_{t>0} \mathcal{OC}^t}$, we conclude that $\frac{1}{m} \sum_{\chi \in \bigcup_{t>0} \mathcal{OC}^t} Y_\chi \leq O(\log n)$ whp, hence $\texttt{Alg}_{\text{TF}}$ sends $O(\ell \cdot \sqrt{n \log n} \cdot \log n)$ messages whp.

▶ **Theorem 10.** *The message load of $\texttt{Alg}_{\text{TF}}$ is $O(\sqrt{n \log n})$ in expectation and $O(\sqrt{n \log n} \cdot \log n)$ whp.*

References

1 Yehuda Afek, Baruch Awerbuch, Serge A. Plotkin, and Michael E. Saks. Local management of a global resource in a communication network. *J. ACM*, 43(1):1–19, 1996. doi:10.1145/227595.227596.

2 Yehuda Afek and Eli Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. *SIAM J. Comput.*, 20(2):376–394, 1991. doi:10.1137/0220023.

3 Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Inf. Comput.*, 113(2):312–330, 1994. doi:10.1006/INCO.1994.1075.

4 Yehuda Afek and Michael E. Saks. Detecting global termination conditions in the face of uncertainty. In Fred B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 109–124. ACM, 1987. doi:10.1145/41840.41850.

5 Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006. doi:10.1137/050645221.

6 Dana Angluin. Local and global properties in networks of processors (extended abstract). In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.

7 Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004.

8 John Augustine, Anisur Rahaman Molla, and Gopal Pandurangan. Sublinear message bounds for randomized agreement. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 315–324. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212751.

9 Baruch Awerbuch, Lenore Cowen, and Mark A. Smith. Efficient asynchronous distributed symmetry breaking. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 214–223. ACM, 1994. doi:10.1145/195058.195136.

10 Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990. doi:10.1145/77600.77618.

11 Reuven Bar-Yehuda, Shay Kutten, Yaron Wolfstahl, and Shmuel Zaks. Making distributed spanning tree algorithms fault-resilient. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 87, 4th Annual Symposium on Theoretical Aspects of Computer Science, Passau, Germany, February 19-21, 1987, Proceedings*, volume 247 of *Lecture Notes in Computer Science*, pages 432–444. Springer, 1987. doi:10.1007/BFB0039625.

12 Bonnie Bassler. *Quorum Sensing: How Bacteria Communicate*. The Explorer's Guide to Biology, September 2019. 48 pages; https://explorebiology.org/summary/cell-biology/quorum-sensing:-how-bacteria-communicate.

13 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

14 Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984. doi:10.1145/800222.806743.

15 Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 12–26. ACM, 1983. doi:10.1145/800221.806706.

16 Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, Vijay K. Garg, and Yogish Sabharwal. Efficient decentralized algorithms for the distributed trigger counting problem. *Theory Comput. Syst.*, 51(4):447–473, 2012. doi:10.1007/S00224-012-9405-4.

17 Che-Cheng Chang and Jichiang Tsai. Distributed trigger counting algorithms for arbitrary network topology. *Wirel. Commun. Mob. Comput.*, 16(16):2463–2476, 2016. doi:10.1002/WCM.2698.

18 Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965. doi:10.1145/365559.365617.

19 Stefan Dobrev, Rastislav Kralovic, Nicola Santoro, and Wei Shi. Black hole search in asynchronous rings using tokens. In Tiziana Calamoneri, Irene Finocchi, and Giuseppe F. Italiano, editors, *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, volume 3998 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2006. doi:10.1007/11758471_16.

20 Yuval Emek and Amos Korman. Efficient threshold detection in a distributed environment: extended abstract. In Andréa W. Richa and Rachid Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 183–191. ACM, 2010. doi:10.1145/1835698.1835742.

21 Yuval Emek and Amos Korman. New bounds for the controller problem. *Distributed Comput.*, 24(3-4):177–186, 2011. doi:10.1007/S00446-010-0119-Z.

22 Yuval Emek, Shay Kutten, Ido Rafael, and Gadi Taubenfeld. Team formation and applications, 2025. doi:10.48550/arXiv.2508.13084.

23 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344. ACM, 2016. doi:10.1145/2897518.2897557.

24 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

25 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. doi:10.1016/J.TCS.2005.01.001.

26 Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/NET.20127.

27 W. Claiborne Fuqua, Stephen C. Winans, and E. Peter Greenberg. Quorum sensing in bacteria: the LuxR-LuxI family of cell density-responsive transcriptional regulators. *Journal of Bacteriology*, 176(2):269–275, 1994.

28 Rahul Garg, Vijay K. Garg, and Yogish Sabharwal. Scalable algorithms for global snapshots in distributed systems. In Gregory K. Egan and Yoichi Muraoka, editors, *Proceedings of the 20th Annual International Conference on Supercomputing, ICS 2006, Cairns, Queensland, Australia, June 28 - July 01, 2006*, pages 269–277. ACM, 2006. doi:10.1145/1183401.1183439.

29 Seth Gilbert and Dariusz R. Kowalski. Distributed agreement with optimal communication complexity. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 965–977. SIAM, 2010. doi:10.1137/1.9781611973075.78.

30 Sergiu Hart. Games in extensive and strategic forms. In *Handbook of Game Theory with Economic Applications*, volume 1, pages 19–40. Elsevier, 1992. doi:10.1016/S1574-0005(05)80005-0.

31 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Comput.*, 28(1):31–53, 2015. doi:10.1007/S00446-013-0202-3.

32 Ling Huang, Minos N. Garofalakis, Anthony D. Joseph, and Nina Taft. Communication-efficient tracking of distributed cumulative triggers. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*, page 54. IEEE Computer Society, 2007. doi:10.1109/ICDCS.2007.93.

33 Pierre A. Humblet. *Selecting a leader in a clique in 0 (N log N) messages*. Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1984.

34 Alon Itai, Shay Kutten, Yaron Wolfstahl, and Shmuel Zaks. Optimal distributed t-resilient election in complete networks. *IEEE Trans. Software Eng.*, 16(4):415–420, 1990. doi:10.1109/32.54293.

35 Prasad Jayanti, Siddhartha Jayanti, and Sucharita Jayanti. Towards an ideal queue lock. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *ICDCN 2020: 21st International Conference on Distributed Computing and Networking, Kolkata, India, January 4-7, 2020*, pages 19:1–19:10. ACM, 2020. doi:10.1145/3369740.3369784.

36 Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 289–300. ACM, 2006. doi:10.1145/1142473.1142507.

37 Seokhyun Kim, Giorgia Fattori, and Yongsu Park. A simple and efficient distributed trigger counting algorithm based on local thresholds. *ICT Express*, 10(4):895–901, 2024. doi:10.1016/J.ICTE.2024.05.005.

38 Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 199–207. ACM, 1984. doi:10.1145/800222.806747.

39 Amos Korman and Shay Kutten. Controller and estimator for dynamic networks. In Indranil Gupta and Roger Wattenhofer, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 175–184. ACM, 2007. doi:10.1145/1281100.1281127.

40 Manish Kumar and Anisur Rahaman Molla. On the message complexity of fault-tolerant computation: Leader election and agreement. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 294–295. ACM, 2023. doi:10.1145/3571306.3571424.

41 Manish Kumar and Anisur Rahaman Molla. Sublinear message bounds of authenticated implicit byzantine agreement. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN 2024, Chennai, India, January 4-7, 2024*, pages 124–133. ACM, 2024. doi:10.1145/3631461.3631548.

42 Shay Kutten. Optimal fault-tolerant distributed construction of a spanning forest. *Inf. Process. Lett.*, 27(6):299–307, 1988. doi:10.1016/0020-0190(88)90217-7.

43 Shay Kutten, William K. Moses Jr., Gopal Pandurangan, and David Peleg. Singularly near optimal leader election in asynchronous networks. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 27:1–27:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.27.

44 Shay Kutten, William K Moses Jr, Gopal Pandurangan, and David Peleg. Singularly optimal randomized leader election. In *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179, 2020. doi:10.4230/LIPICS.DISC.2020.22.

45 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theor. Comput. Sci.*, 561:134–143, 2015. doi:10.1016/J.TCS.2014.02.009.

46 Shay Kutten, Peter Robinson, Ming Ming Tan, and Xianbin Zhu. Improved tradeoffs for leader election. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 355–365. ACM, 2023. doi:10.1145/3583668.3594576.

47 Leslie Lamport. A new solution of dijkstra's concurrent programming problem. *Commun. ACM*, 17(8):453–455, 1974. doi:10.1145/361082.361093.

48 Giuseppe Antonio Di Luna, Ryuhei Uehara, Giovanni Viglietta, and Yukiko Yamauchi. Gathering on a circle with limited visibility by anonymous oblivious robots. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.12.

49 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

50 Earl H. McKinney. Generalized birthday problem. *The American Mathematical Monthly*, 73(4):385–387, 1966.

51 El Mahdi El Mhamdi, Rachid Guerraoui, Alexandre Maurer, and Vladislav Tempez. Exploring the borderlands of the gathering problem. *Bull. EATCS*, 129, 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/593/602.

52 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

53 Stephen C. Pratt. Quorum sensing by encounter rates in the ant Temnothorax albipennis. *Behavioral Ecology*, 16(2):488–496, 2005.

54 Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.48.

55 Red Hat Publication. Stateful vs. stateless applications. Accessed online, February 5th, 2025. URL: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless#:~:text=Stateless%20applications%20are%20generally%20more,it%20more%20difficult%20to%20scale.

56 Thomas D. Seeley and P. Kirk Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–16, 2004.

57 Eugene Styer and Gary L. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 177–191. ACM, 1989. doi:10.1145/72981.72993.

58 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

59 Gadi Taubenfeld. *Synchronization Algorithms and Concurrent Programming*. Pearson / Prentice-Hall, 2006. ISBN 0-131-97259-6, 423 pages.

60 Gadi Taubenfeld, Shmuel Katz, and Shlomo Moran. Initial failures in distributed computations. *Int. J. Parallel Program.*, 18(4):255–276, 1989. doi:10.1007/BF01407859.

61 Masafumi Yamashita and Tiko Kameda. Computing on an anonymous network. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 117–130. ACM, 1988. doi:10.1145/62546.62568.

Appendix

A Applications – Additional Information

In this section, we provide further technical details of some of the applications discussed in Sec. 1.4. The details of the other applications are omitted from this extended abstract and can be found in the full version [22].

A.1 Leader Election

A.1.1 Implicit leader election

Implicit LE is defined in Sec. 1.4. We first derive a simpler algorithm $LE-I1$ that has no termination detection for some nodes and add the termination detection later.

Informally, it is assumed that there are $(1/2+\epsilon)n$ non-fragile nodes and each of them wakes up and starts the algorithm. Any fragile node may also be set by the adversary to be non-faulty at that point and start the algorithm, too.

1. Each node starting the algorithm flips a coin with probability $\frac{c\log n}{n}$ (for some constant c that is "large enough"; the selection of c uses rather standard considerations, so we skip it here). Let $Cand$ be the set of nodes that flipped 1.
2. Each node not in $Cand$ sets its status to "not-leader."
3. Each node in $Cand$ generates a token and starts the algorithm for TF with $\sigma = (1-\frac{\epsilon}{2})(\frac{1}{2}+\epsilon)c\log n$ (rounded up, let us ignore that for simplicity of notation).
4. A node that deleted σ tokens in TF sets its status to "leader."

The following observation follows directly from Thm. 1 and the algorithm. (For the second part, note that $\sigma = O(\log n)$.)

▶ **Observation 11.** *Algorithm $LE-I1$ satisfies the following two properties:*

- *A node that is not the elected leader may not know that it is not the leader. (We shall fix this later.)*
- *The message complexity of Algorithm $LE-I1$ is $O(\sqrt{n\log n})$ in expectation and $O(\sqrt{n\log n}\cdot\log n)$ whp. The time complexity is $O(\log n)$.*

▶ **Lemma 12.** *Algorithm $LE-I1$ elects a unique leader whp.*

Proof. Let us analyze the relation between σ and the number T of tokens generated by nodes in the third line of the algorithm. The number n' of nodes that flipped coins may include any number (between zero and $(1/2-\epsilon)n$) of fragile nodes; hence, $(1/2+\epsilon)n \le n' \le n$. Thus, for the expected number $\frac{cn'\log n}{n}$ of tokens generated, we know that $(1/2+\epsilon)c\log n \le \frac{cn'\log n}{n} \le c\log n$. Applying Chernoff's bound, we find for the random variable T (the number of tokens) that whp,

$$(1-\frac{\epsilon}{2})(\frac{1}{2}+\epsilon)c\log n \le T \le (1+\frac{\epsilon}{2})c\log n.$$

By the left inequality, $\sigma \le T$ whp. Hence, at least one set of tokens can be deleted in TF, which means that at least one node is elected in step 4 of the algorithm.

It remains to show that $T < 2\sigma$, so no more than one deletion event occurs (whp) and thus, no more than one leader is elected. By the above bounds on the number of tokens, $T/\sigma = \frac{T}{(1-\epsilon/2)(1/2+\epsilon)c\log n} \le \frac{(1+\epsilon/2)c\log n}{(1-\epsilon/2)(1/2+\epsilon)c\log n} \le \frac{(1+\epsilon/2)}{(1-\epsilon/2)(1/2+\epsilon)}$. This is strictly smaller than 2. ◀

Termination Detection for Non-Leaders. Let us now show how to enhance algorithm $LE-I1$ such that every node that is *not* elected will set its status to "not-leader" eventually, as is sometimes required. That is, the promised $LE-I$ algorithm will first run Algorithm $LE-I1$ (that runs the TF algorithm). After that, the elected leader will invoke a termination detection module $Term$.

We came up with several methods of implementing $Term$. Let us describe two different implementations. The first one uses the accumulation property of the algorithm (see

Sec. 1.3). The second one does not use the accumulation property and is given here to show that a reduction from LE to TF is possible without using additional features. For both implementations, mark every token created in the third step of $LE - I1$ by "LE token". The implementations use an additional kind of token, called TERM (described below). The TF algorithm (run as a subroutine of the LE algorithm) does not distinguish between the two kinds of tokens. However, the $Term$ part of the LE notices when a TERM token participates in a formation (second implementation) or is received (first implementation) and makes use of this knowledge.

First Implementation. The leader r (the node who performed the team formation) creates one TERM token. The underlying TF algorithm treats the TERM token exactly like any other token. However, any non-leader node $v \neq r$ performing $LE - I1$ that receives a TERM token, note that it (v) is not a leader.

Recall that by the accumulation feature of the algorithm, all the tokens that remain in the networks after the team formation eventually are held (forever) by a single node v. If $v \neq r$, then v knows it is not the leader. Let w be any node other than v and r. Either w did not join $Cand$, or it did join but at some point sent its token to another node. In both cases, w knows that it is not the leader. It is easy to show that this implementation does not increase the order of the complexities of the LE election algorithm beyond that of $LE - I1$.

Second Implementation. In this implementation of $Term$, when the leader node deletes σ tokens, it generates $\sigma - 1$ TERM tokens and continues to perform TF. Any other node that deletes σ tokens that include some x TERM tokens and some $\sigma - x$ LE ones generates x TERM ones. Any node that is *not* the leader but ever sees a TERM token, changes its status to "not-leader." The arguments for correctness are similar to those for the first implementation. Note that introducing those new tokens increases the complexities of the algorithm by a logarithmic factor.

A.1.2 Explicit LE

So far, a non-leader node may not know which of its edges leads to the leader. To make the algorithm solve explicit leader election, the elected leader r sends each of its neighbors one message, notifying the neighbor that r is the leader. This adds n messages and a single time unit to the complexities of the election.

B Additional Related Work

The problem of reaching consensus in a complete graph with initial faults was introduced in the seminal paper of [24] since they showed that reaching consensus was impossible in complete asynchronous networks when faults could occur at any time. The same algorithm could be used for leader election. Its message complexity in the CONGEST model was $O(n^3)$ if $O(n)$ nodes could fail. This was improved to $O(n^2)$ in [11] that also addressed general graphs and [34] that solved leader election (and thus also consensus) in a complete graph where the algorithm is given $k < n/2$ that is the maximum possible number of initial faults. The message complexity was $O(n \log n + nk))$ which is $O(n^2)$ in complete networks. For general graphs, a further improvement was obtained in [4] by combining their counting algorithm with an algorithm for election (with no termination detection) in the presence of initial faults, such as the algorithm of [42]. Their message complexity was $O(|E| + n \log^5 n)$. The above mentioned counting was improved in [1], implying a message complexity of

$O(|E| + n \log^3 n)$ for deterministic leader election and consensus under initial faults in general graphs. This means $O(n^2)$ in complete graphs.

Randomization was used to enable consensus in a complete graph in case nodes could fail at any time. Various algorithms were introduced, with various promises on resilience (e.g., also addressing Byzantine faults, varying the number of faults), time complexity, probabilistic assumptions, etc. [13, 15, 14, 54]. They send $\Omega(n^2)$ messages.

Election and consensus in *non-faulty* complete graphs have been studied extensively; see, e.g., [38, 2, 33, 3, 43, 46]. The message complexity in most of them is $\Omega(n \log n)$, with some being $O(n)$, e.g., in the randomized synchronous case. Surprisingly, $o(n)$ (specifically, $O(\log n\sqrt{n})$ was eventually shown for synchronous fault-free networks for the implicit version of the problem (that is, not every node needs to know the output, but no two outputs contradict) [45]. This has been generalized to algorithms that can withstand crash faults [40] or even byzantine faults (assuming authentication) [41] and sublinear message complexity of agreement was also studied [8] but all these was still done in *synchronous* networks. The birthday paradox [50] is used in the above papers; it is used here too, for building a more general structure that supports more general results (such as asynchrony and initial failures).

The *explicit* synchronous case of consensus with crash faults was addressed in [29], using $\Theta(n)$ messages and $\Theta(\log n)$ time.

For the *asynchronous* fault-free case, too, $\Omega(n)$ was considered to be optimal (see, e.g. [43]). It may indeed be for the explicit case, but recall that one of the results in the current paper is an $o(n)$ message result for the *asynchronous* case, even for networks with (generalized) initial faults. The time complexity in [44] for the *non-faulty* synchronous case (and in [43] for the *non-faulty* asynchronous case) is $O(\log^2 n)$, improved here to $O(\log n)$ with at least as good as those of [43, 44] (the message complexity as good as that of the previous synchronous solution and potentially better than the previous asynchronous one).[15]

Various papers addressed problems that are variations of the application we call here distributed trigger counting (DTC). (Some such variations are called threshold detection, threshold sensing, and controller.) The one-shot case is the one where the number of triggers (called "events" in some other papers) passes some threshold [1, 39, 28, 16, 32, 21]. The message complexity obtained was not always analyzed (in the more practical papers), but when analyzed, it was at least $O(\log n \log \sigma)$ even in the case of complete graphs [28]. In the latter case a message could be sent directly from a site (node) to a leader (a coordinator node) that is known in advance. (That model allows sending messages between sites, but the algorithms do not use that.)

The ongoing case is treated, e.g., in [36]. The message complexity per counted event is about the same as in the one-shot case, and some errors in the counting are allowed.

The notion of forgetful algorithms extends the notion of *memoryless* algorithms first defined for deterministic shared-memory mutual exclusion algorithms in [57]. A memoryless mutual exclusion algorithm is an algorithm in which, when all the processes are in their remainder sections, the registers' values are the same as their initial values. This means that a process that tried to enter its critical section did not use any information about its previous attempts (such as the fact that it has entered its critical section five times so far). Almost all known mutual exclusion algorithms are memoryless [59], including Lamport's famous Bakery algorithm [47].

[15] The previous result for the asynchronous case was for general graphs.

C Discussion

We introduce and study the TF problem and demonstrate its usefulness. We consider an asynchronous message-passing system assuming a complete network topology. It would be interesting to study TF in other models of computation assuming, for example, a more general network topology, synchronous systems, a situation where the nodes and/or the tokens are not anonymous, assume that the nodes know the identities of their neighbors (i.e., the KT_1 model [10]), changing the assumption about the message size, etc.

Even for complete networks, interesting problems remain. Can the lower bound be improved? The TF solution helped us improve the known results for some applications. Does it mean that the complexities of these applications are inherently similar to that of TF? or are there upper bounds for some of the applications better than the lower bound for TF? We addressed the case of a small team size σ that seems more practical and challenging; how about the case of a large σ? The complexity per token in the case of a large σ may be much lower. For example, if $\sigma = n + 1$, no team formation is possible unless some node gets at least 2 tokens injected. Hence, it makes sense that an algorithm at a node will do nothing as long as it holds a smaller number of tokens (until approached by another node).

We generalized the TF problem to address a team represented by vectors such as x_1 tokens of color red *and* x_2 tokens of color blue (e.g., x_1 sorcerers and x_2 warriors). How about using other logical operations when composing teams? For example, x_1 sorcerers *and* either x_2 warriors *or* x_3 thieves?

Our algorithms have several novel properties. These properties may be useful for studying other problems, not just TF. One of the properties is, in the context of long-lived problems, resilience to failures *stronger* than initial failures. This might be useful for other long-lived problems. It would be interesting to compare the power of the model that allows such failures to the power of other failure models.

Another important notion we introduced in the context of randomized algorithms is that of a *forgetful* algorithm. While, for deterministic algorithms, this notion is the same as that of a memoryless algorithm, for randomized algorithms, this notion is not as strong as one may expect, as we assume that a node will never forget its initial coin tosses. We use this slightly limited version to ensure our algorithms succeed with high probability, even for infinite executions. Had we sufficed with high probability for each team formation separately, then we would have been able to remove this limitation and ensure that a node does not remember anything (not even its initial coin tosses) in the TF algorithm.[16]

Requiring that nodes forget their past behavior in some cases usually leads to simple solutions, saves the cost of maintaining data structures during quiescent times, and eases the recovery in case of catastrophes, since no data structure needs to be recovered. We emphasize again that in long-lived problems (like mutual exclusion and team formation), a naive solution where a leader is elected and forever coordinates all activities is not forgetful (or memoryless), even in fault-free models.

Let us note that various related notions appeared in the literature. For example, notions of "not keeping information about past behavior" are promoted in practice, e.g., [55], and theory, e.g., [35], because they are considered more scalable and easier to recover.

An interesting property of our TF algorithm seems to be related to privacy and requires further discussion (we have not analyzed this property formally). Even in anonymous networks, nodes in many algorithms (but not the algorithm of this paper) choose identities

[16] When allowing a node to fully forget everything, in an infinite number of team formation where each succeeds whp separately, some formations might not succeed – our slightly limited notion of forgetfulness prevents this situation from happening.

that are unique whp. Suppose that node u sends message M_1 to node v and messages M_2 to node w. Nodes v and w may now compare M_1 and M_2 knowing that they both originated from u. This may harm u's privacy. For example, u may not want other nodes to know that its token participated in a team since this team has some private purpose. Let us say that the algorithm that has this property is *inconspicuous*. It would be interesting to define it formally, prove formally that our algorithm satisfies this property, and study the power of models that allow only such algorithms.

Our generalization of initial failures is not weaker than crash failures. When there are only crash failures, in an infinite execution, a node either never crashes or it is alive only for a finite time (the finite prefix before it crashes). This claim is not valid with our "fragile failures" since there is no limit on how many times the status of a node can be toggled. Another observation regarding crash failures is that TF is not solvable, even with randomization, in the presence of a single crash failure. This observation follows from the fact that a node may crash while holding a token or just before a token is sent to it. (This impossibility holds even with the restriction that nodes with tokens may not crash.) It is possible to modify the liveness condition in the definition of TF so that the above argument does not apply.

Finally, we point out an extensively studied biological phenomenon similar to the one captured by TF. *Quorum sensing* is a process in which bacteria can sense that the number of bacteria ready to release toxins (which cause disease in plants, animals, and humans) has reached a certain threshold. This enables them to release the toxins at approximately the same time [12, 27]. Several insects, like ants and honey bees, have been shown to also use quorum sensing in a process that resembles collective decision-making [53, 56]. Unlike the trigger counting problem (mentioned in the Introduction), here, most participants must know when the threshold is reached.