Stabilizing Consensus in Mobile Networks

Dana Angluin[*] Michael J. Fischer[*] Hong Jiang[*†]

February 15, 2006

Abstract

Inspired by the characteristics of biologically-motivated systems consisting of autonomous agents, we define the notion of stabilizing consensus in fully decentralized and highly dynamic ad hoc systems. Stabilizing consensus requires non-faulty nodes to eventually agree on one of their inputs, but individual nodes do not necessarily know when agreement is reached. First we show that, similar to the original consensus problem in the synchronous model, there exist deterministic solutions to the stabilizing consensus problem tolerating crash faults. Similarly, stabilizing consensus can also be solved deterministically in presence of Byzantine faults with the assumption that $n > 3f$ where n is the number of nodes and f is the number of faulty nodes. Our main result is a Byzantine consensus protocol in a model in which the input to each node can change finitely many times during execution and eventually stabilizes. Finally we present an impossibility result for stabilizing consensus in systems of identical nodes.

1 Introduction

1.1 Fault-tolerant consensus

Coordination problems in distributed systems require nodes to agree on a common action. Lamport, Pease, and Shostak formulated this problem as the agreement problem [22, 25], which remains a fundamental problem in distributed computing. It is usually trivial to reach agreement in reliable systems. In practice, however, different components in a system don't always work correctly. Mission-critical control systems require agreement among non-faulty components even when some components are faulty. The problem was originally defined for Byzantine faults, which means faulty nodes in a network could behave arbitrarily. A more benign type of faults is crash faults, meaning a faulty node may stop all activities at a certain point in the execution. Sometimes the recovery of crashed processes is also considered. [22, 25] gave a synchronous f-resilient solution for any f with authentication in the case of a complete communication graph and the impossibility result that consensus is not solvable without authentication unless the number of faulty processes is less than one-third of the total. Dolev [11] considered the Byzantine agreement problem in networks that are not completely connected. The first polynomial communication algorithm

[*] Department of Computer Science, Yale University.

[†] Supported by NSF grant ITR-0331548

for Byzantine agreement was designed by Dolev and Strong [14], whose work was subsequently improved by Dolev, Fischer, Fowler, Lynch, and Strong [12]. Fischer, Lynch, and Paterson [20] showed that in a fully asynchronous environment, there is no 1-resilient solution to the consensus problem, even for crash failure. A survey on fault-tolerant consensus by Fischer [18] provides an overview of early work on fault-tolerant distributed systems.

One of the reasons for the impossibility results for fault tolerance in the asynchronous model is that, on one hand, messages are allowed to be delayed arbitrarily as long as they are eventually delivered, and therefore there is no way to distinguish between a crashed node and a slow node, or a lost message and a delayed message; on the other hand, distributed algorithms are expected to terminate, and each non-faulty process is required to *commit* to an output at some point in its execution when it knows the decision that all non-faulty processes will agree on.

1.2 Motivation

In some persistent ad hoc networks, especially biologically-motivated systems, it is not important that each process be aware of the global status. For example, imagine the aggregation and migration of birds. During the initial gathering or a direction change, the movement of the birds is usually chaotic. Although each bird is not aware of the status of the whole flock, the flock eventually converges to a stable state in which all the birds head in roughly the same direction. Each bird adjusts its heading according to what it perceives but does not commit to a direction at any point, because it is possible that some other birds are still changing directions.

Vicsek *et al.* [29] described a compelling model of dynamics in order to investigate the emergence of self-ordered motion in systems of autonomous agents with biologically motivated interaction. Each agent's heading is updated from time to time according to a local rule. They demonstrated that all agents eventually move in the same direction despite the absence of centralized coordination and the changing neighbor set of each agent as the system evolves. Agents do not know accurately when the whole system converges, except for an estimation of expected convergence time when the agents communicate synchronously and certain topological properties are guaranteed as the network evolves. Jadbabaie, Lin, and Morse [21] provided a theoretical explanation for the above behavior and investigated other similarly inspired models. Agent failures were not considered in these two papers.

We discuss asynchronous fault-tolerant consensus in a similar scenario. We relax the requirement of the original consensus problem: in particular, agents do not need to know whether a decision is final. We also investigate persistent distributed systems that run for an extended period and whose inputs may change from time to time, and systems that have incomplete or evolving interaction graphs. We also prove that stabilizing Byzantine consensus cannot be solved in systems consisting of identical nodes.

1.3 Other related work

Because agreement is a fundamental problem in building distributed systems, and most practical systems are not synchronous, various methods have been used to circumvent the impossibility result in [20]. In many practical systems, nodes periodically send "I'm alive" messages (pings) to each other to detect possible crashes. In theoretical models, such techniques are captured by the abstract concept of failure detectors [9]. Generally, a failure detector is a module that provides information to processes about previous failures. Failure detectors differ in strength depending on whether they are always correct or whether they detect all failures [8]. Some failure detectors can be implemented in practical systems using timeouts.

The agreement problem can be solved in a randomized asynchronous model which allows each process to flip coins during execution. The problem statement is modified to require that the processes eventually terminate with probability 1. The first randomized solution for consensus was given by Ben-Or [4]. Rabin [27] and Feldman [17] produced more efficient algorithms.

The k-agreement problem [10] is a weakened problem statement that only requires all the decisions to be in a set of k values. Another weakened variation is the approximate agreement problem [13] which allows inputs, decisions, and messages to be real numbers and requires the difference between any two decision values to be within a small tolerance ϵ and that any decision value is within the range of input values.

Lamport invented the PAXOS algorithm [23] for a partially synchronous model of distributed systems. Asynchrony is considered timing failure. Other failures allowed are loss, duplication and reordering of messages, and crash failure of processes. Process recovery is also allowed. The PAXOS algorithm guarantees safety, meaning that in spite of timing, process, and link failures, and process recoveries, non-faulty nodes do not decide on inconsistent values. When the system stabilizes (no failure occurs, and a majority of the processes are active) for a sufficiently long time, termination can also be achieved.

Although mobile computing has been under active study for many years, research on fault-tolerance issues has been limited, and many desirable goals are yet to be achieved. The problems studied include ad hoc routing [5,28], which allow nodes to exchange data despite the limited transmission range of wireless interfaces by routing messages through multiple network hops, broadcasting and multicasting [31], transaction control [2, 6, 26], group communication [7, 16], leader election [24], and mutual exclusion [30]. Angluin *et al.* [1] proposed self-stabilizing solutions to problems like leader election, ring orientation, token circulation, and spanning-tree construction in a model of pairwise interacting anonymous finite-state sensors under a global fairness condition. Basile, Killijian, and Powell [3] gave a survey of fault tolerance in mobile wireless networks. The fault models considered in existing works are usually mobility, network partitioning, and sometimes crash failures. Byzantine node failures have generally not been considered.

2 Model and definitions

We consider a network of n mobile nodes. Each has a unique ID $\in [1, n]$, an input port, and an output port to send outputs to an external observer. By assigning unique IDs to the nodes, we give each node the ability to distinguish between different nodes. Also, the IDs are assumed to be unforgeable in the sense that a faulty node cannot impersonate a non-faulty node in direct communications (i.e. messages not involving any intermediate nodes). The nodes do not have access to timing features like global or local clocks, pulses, or time-out notification. Nodes can communicate with each other by sending messages. A node i can send a message to another node j when j is close enough to i. Messages may get lost, but we assume that if i sends a message to j

infinitely often, j receives the message infinitely often. We also assume the fairness condition that every pair of nodes have a chance to communicate (move close enough to send messages) infinitely often. We remark that this model is weaker than the asynchronous model of distributed systems in [20], therefore one cannot expect to solve the original consensus problem in presence of faults.

In the literature two kinds of failures (equivalently, faults) are usually considered for the consensus problem. The benign type of failure is crash failure: a faulty node may crash at anytime. When a node crashes, it stops operating but does not do any wrong operation, otherwise it honestly follows the protocol. We assume that if a node crashes when sending a message, the incomplete message is discarded by the recipient. Byzantine failures are more severe. A Byzantine node may behave arbitrarily without the limit of computational power or memory usage to which a non-faulty node is constrained otherwise.

We define the notion of *stabilizing consensus*. Instead of requiring that each node commit to a final output at some point, we assume each node has a current output which may change as the execution proceeds. In practical applications, the output could be interpreted as some parameter that reflects the behavior of each node. For example, in a flock of mobile nodes, the current output of each node could be its current speed or current direction.

As in the usual convention, a configuration includes all nodes' local states and the pending messages. A configuration C is said to be *output-stable* if in all possible executions starting from C, the output of each non-faulty node does not change. If every non-faulty node outputs x in an output-stable configuration C, we say the outputs stabilize to x in C.

Definition 1 (Stabilizing Consensus)**.** A protocol P solves the stabilizing consensus problem if all of the following requirements are satisfied:

Stabilization The system eventually reaches an output-stable configuration.

Validity The requirement depends on the fault model:

> **Crash-fault version** If all nodes have the same input x, the outputs of all non-faulty nodes eventually stabilize to x.
>
> **Byzantine version** If all non-faulty nodes have the same input x, the outputs of all non-faulty nodes eventually stabilize to x.

Agreement In any reachable output-stable configuration, all non-faulty nodes have the same output.

In the following sections, we consider both crash faults and Byzantine faults. We also discuss consensus in a scenario where each node receives an input that may change finitely many times and define *consensus with stabilizing inputs*. Finally we show that stabilizing consensus cannot be solved in a system consisting of identical nodes in the presence of one Byzantine fault.

3 Stabilizing consensus with crash faults

The following is a simple protocol that solves consensus in the presence of crash faults, assuming the inputs are non-negative integers. The protocol is a straight-forward adaptation of the protocol for synchronous distributed systems from [15].

For each node i, x_i is its local input (a non-negative integer), and y_i is its output.

- At the beginning, node i sets $y_i = x_i$.
- Whenever i is able to send a message, it sends y_i to the recipient.
- Upon receiving message y_j from some node j:

  $y_i = \min(y_i, y_j)$

Notice that in general, a node does not know when its output stabilizes.

The following theorem establishes the correctness of the protocol.

Theorem 2. *The above protocol solves stabilizing consensus in presence of f crash faults for any $f < n$, where n is the total number of nodes.*

Proof sketch:

It is easy to see that the outputs will stabilize, because the output of each node can only decrease and cannot be negative. Also it is clear that the validity condition is satisfied because if all nodes have the same input, they will all receive the same messages and output that same value.

Suppose for the sake of contradiction that two nodes i and j stabilize to different outputs, y_i and y_j. Without loss of generality, we assume $y_i < y_j$. According to the fairness condition, eventually j will receive a value of y_i from i and set $y_j = \min(y_i, y_j) = y_i$, contradicting the assumption that the output of j stabilized to y_j.

4 Stabilizing consensus with Byzantine faults

4.1 A protocol for fixed inputs

In this section we give protocols tolerating Byzantine faults. We assume that when a node receives a message, it knows the identity of the sender. We first consider a system where each node receives an input at the beginning. Each node i has a fixed local input x_i. For simplicity, we assume $x_i \in \{0, 1\}$. We give a protocol that tolerates f Byzantine faults, assuming $3f < n$ where n is the total number of nodes.

The state of each non-faulty node i consists of the arrays $I_i[n]$, $E_i[n][n]$ and $M_i[n]$, in which all elements are initialized to 0.

- When node i is able to send a message, it sends a message including one or more of the following components:

  If $x_i = 1$, it sends (init, i) to the recipient.

  For all j such that $I_i[j] = 1$ or $\sum_{k=1}^{n} E_i[j][k] \geq f+1$, i sends (echo, j) to the recipient.

- When i receives (init, j) from j, it sets $I_i[j] = 1$.
- When i receives (echo, k) from j, it set $E_i[k][j] = 1$, and if $\sum_{k=1}^{n} E_i[j][k] \geq n - f$, i sets $M_i[j] = 1$

Output:
The current output of node i is 1 if $\sum_j M_i[j] \geq 2f + 1$, otherwise its output is 0.

It is easy to see that the outputs will stabilize, because each node outputs 0 initially, and can flip its output to 1 at most once.

Correctness can be established by verifying the following claims.

Lemma 3. *If any non-faulty node i has 1 as input, eventually every non-faulty node j sets $M_j[i] = 1$.*

Eventually every node receives (init, i) from node i, and all non-faulty nodes will send (echo, i) to every other node it encounters. Therefore any non-faulty node j will receive (echo, i) from at least $n - f$ nodes and set $M_j[i] = 1$

Lemma 4. *If any non-faulty node i has 0 as input, $M_j[i]$ is always 0 for any non-faulty node j.*

In this case, no non-faulty node receives (init, i) from node i. Suppose node j is the first non-faulty node that sends (echo, i). It must have been triggered by receiving (echo, i) from $f+1$ faulty nodes, which contradicts the assumption. A non-faulty node j never sends (echo, i) and receives (echo, i) from at most f faulty nodes, so it will never set $M_j[i] = 1$.

Lemma 5. *For any i, if $M_j[i]$ stabilizes to 1 in any non-faulty node j, $M_k[i]$ eventually stabilizes to 1 in any other non-faulty node k.*

If any non-faulty node j sets $M_j[i] = 1$, it must have received (echo, i) from at least $n - f$ nodes among which there are at least $f + 1$ non-faulty nodes. The messages (echo, i) sent by these $f + 1$ nodes are received by all non-faulty nodes, therefore all non-faulty nodes will send (echo, i) to each non-faulty node k so that it sets $M_k[i] = 1$.

Theorem 6. *The above protocol solves the stabilizing consensus problem.*

Given the above claims, it is easy to see that the protocol satisfies stabilization, validity, and agreement.

4.2 Stabilizing inputs

We define a model of stabilizing inputs to a network protocol in which the input to each node may change finitely many times before it stabilizes to a final value. We are interested in solving the consensus problem corresponding to the final stabilized input assignment. This consistent input and output convention makes a solution suitable as middleware in constructing more complex systems. Here we define what consensus means in this model.

Definition 7 (Consensus with Stabilizing Inputs)**.** A protocol P solves consensus with stabilizing inputs if all of the following requirements are satisfied:

Stabilization If the inputs to the non-faulty nodes stabilize, the system eventually reaches an output-stable configuration.

Validity If all non-faulty nodes have the same stabilized input x, their outputs eventually stabilize to x.

Agreement In any reachable output-stable configuration, all non-faulty nodes have the same output.

Fixed inputs is a special case of stabilizing inputs.

The following protocol achieves consensus with stabilizing inputs tolerating f Byzantine faults, assuming $3f < n$ where n is the total number of nodes. Here we give only a high-level description of the protocol, because our purpose is to establish the possibility of a protocol, rather than an optimal implementation. In our description, each node needs to keep track of messages received in the past. This intensive memory usage could be reduced by garbage-collecting data that is no longer useful in subsequent computation. We postpone the details of implementation and optimization to the full version of the paper.

Each non-faulty node i maintains two arrays $M_i[n]$ and $C_i[n]$. The elements of M_i are initialized to 0, and the elements of C_i are initialized to -1. It also has a counter c_i initially equal to 0. Let $x_i \in \{0, 1\}$ denote the current reading of the input port. Node i also maintains a variable x'_i and initially sets $x'_i = x_i$.

- When i is able to send a message:
  1. If $x_i \neq x'_i$, set $x'_i = x_i$ and $c_i = c_i + 1$;
  2. Always send (init, i, x_i, c_i);
  3. For all j, x_j, and c_j, such that i has received (init, j, x_j, c_j) from j, or i has received (echo, j, x_j, c_j) from at least $f+1$ different nodes, send (echo, j, x_j, c_j) to the recipient.
- When i receives (init, j, x_j, c_j) from j, if $c_j \leq C_i[j]$, the message is ignored, otherwise it records this message in its event log. If i receives contradicting init messages from the same node ((init, j, x_j, c_j) and (init, j, x'_j, c_j) with $x_j \neq x'_j$), only the first message is recorded.
- When i receives (echo, j, x_j, c_j), if $c_j \leq C_i[j]$, the message is ignored, otherwise it records this message in its event log, and if the same message has been received from at least $n - f$ different nodes, i sets $M_i[j] = x_j$ and $C_i[j] = c_j$

Output:

- Define the *stable set* S_i to be a set of $2f + 1$ distinct integers in $[1 \ldots n]$ that minimizes $\sum_{j \in S_i} C_i[j]$. In case of ties, the set that minimizes $\sum_{x \in S_i} x$ is chosen.
- Node i outputs 1 if $\sum_{j \in S_i} M_i[j] \geq f + 1$, otherwise it outputs 0.

The variable c_i is a counter for node i to keep track of how many times its input has changed. Each node also uses the counter array C_i to keep track of the number of times the other nodes change their inputs. Because messages can be delivered out of order, and "echo" messages corresponding to inputs at different time can co-exist in the network, the counters also ensure that obsolete messages are ignored.

Lemma 8. *The invariant $C_i[j] \leq c_j$ holds in any real-time snapshot of the system for any non-faulty nodes i and j.*

Proof. Suppose at some point in real time, $C_i[j] = a$, $c_j = b$ and $a > b$. Then i must have received (echo, j, m, a) for some m from at least $n - f$ nodes. Therefore j must have sent (init, j, m, a), because at most f nodes send (echo, j, m, a) otherwise. This contradicts $a > b$. Because j would have set $c_j = a$ before sending (init, j, m,a), it must be true that $b \geq a$. □

Lemma 9. *Let i and j be two non-faulty nodes. If i's input stabilizes to x, $M_j[i]$ eventually stabilizes to x.*

Proof. Suppose $M_j[i]$ stabilizes to $y \neq x$. Then j must have received (echo, i, y, a) for some a from at least $n - f$ nodes, so i must have sent (init, i, y, a) to at least $f + 1$ nodes. Since x is the final input of i, eventually i sends (init, i, x, b) for some b to all nodes. According to lemma 8 $a < b$. Suppose the time j receives (echo, i, y, a) from the $(n - f)$th node is t, and the time it receives (echo, i, x, b) from the $(n - f)$th node is t'. If $t < t'$, j will set $M_j[i] = x$. If $t' < t$, y is ignored by j, because at t the value of $C_j[i]$ can only be greater than or equal to b and $a < b$. Therefore $M_j[i]$ couldn't have stabilized to y. □

Lemma 10. *If i and j are non-faulty nodes, for any k, if $M_i[k]$ stabilizes to x, $M_j[k]$ also stabilizes to x.*

Proof. Suppose $M_i[k]$ stabilizes to x, $M_j[k]$ stabilizes to y, and $x \neq y$. Let (echo, k, x, a) and (echo, k, y, b) be the corresponding messages received by i and j respectively when they assigned the final values to $M_i[k]$ and $M_j[k]$.

1. Without loss of generality, we assume $a > b$. Since i must have received (echo, k, x, a) from at least $n - f$ nodes, there must be at least $f + 1$ non-faulty nodes in them. All non-faulty nodes would receive (echo, k, x, a) from these $f + 1$ nodes, and therefore would send (echo, k, x, a) to all nodes they encounter. Thus j would also receive (echo, k, x, a) from at least $n - f$ nodes. Because $a > b$, $M_j[k]$ could not have stabilized to y.

2. If $a = b$, i receives (echo, k, x, a) from $n - f$ nodes, and j receives (echo, k, y, b) from $n - f$ nodes. This cannot happen, because $n > 3f$, and according to the protocol, a non-faulty node only sends one of the two messages but not both.

Therefore $M_i[k]$ and $M_j[k]$ cannot stabilize to different values for any k. This property guarantees that all non-faulty nodes will eventually agree on the stabilized entries of vector M. □

Lemma 11. *Let i and j be any non-faulty nodes. For any k, if $C_i[k]$ stabilizes to c, $C_j[k]$ also stabilizes to c.*

Proof. Suppose $C_i[k]$ stabilizes to c_1, $C_j[k]$ stabilizes to $c_2 \neq c_1$. Without loss of generality, we assume $c_1 > c_2$. Let (echo, k, x, c_1) and (echo, k, y, c_2) be the corresponding messages received by i and j respectively when they assign the final values of $C_i[k]$ and $C_j[k]$. Since i must have received (echo, k, x, c_1) from at least $n - f$ nodes, there must be at least $f + 1$ non-faulty nodes in them. All non-faulty nodes would receive (echo, k, x, c_1) from these $f + 1$ nodes, and therefore would send (echo, k, x, c_1) to all nodes they encounter. j would also receive (echo, k, x, c_1) from at least $n-f$ nodes. Because $c_1 > c_2$, $C_j[k]$ could not have stabilized to c_2. This property guarantees that all non-faulty nodes will eventually agree on the stabilized entries of vector C. □

Lemma 12. *In any execution of the above protocol, if the inputs to the non-faulty nodes stabilize, the outputs of the non-faulty nodes eventually stabilize.*

Proof. Let i be any non-faulty node. If x_i stabilizes, c_i also stabilizes, because they always change at the same time. According to lemmas 8 and 9, $M_j[i]$ and $C_j[i]$ also stabilize for any non-faulty j. According to lemma 10 and lemma 11, all non-faulty nodes will eventually agree on the stabilized entries of the arrays M and C (at least $2f+1$ entries in each), which include entries corresponding to non-faulty nodes and entries corresponding to faulty nodes that stabilize at all. If some of the entries in the M arrays corresponding to faulty nodes do not stabilize, the corresponding entries in the C arrays of the non-faulty nodes will eventually be greater than the stabilized entries, because the entries of C arrays are non-decreasing. Only the $2f-1$ nodes corresponding to the C entries with the smallest values affect the output, therefore the faulty nodes will eventually be ignored. □

Theorem 13. *The above protocol solves consensus with stabilizing inputs.*

If all non-faulty nodes have $x \in \{0,1\}$ as input, according to lemma 9, for any non-faulty node i at least $f+1$ M_i entries corresponding to the stable set will be x, therefore all non-faulty nodes will output x, and the validity condition is satisfied. According to lemmas 10 , 11, and 12, agreement and stabilization are also satisfied.

5 Impossibility of stabilizing Byzantine consensus among identical nodes

In this section we give the impossibility result that stabilizing consensus cannot be solved in the presence of a single Byzantine fault in a network of nodes that are identical other than their inputs. We note that any subconfiguration of an output-stable configuration is also output-stable.

Theorem 14. *The stabilizing consensus problem cannot be solved in a set of identical nodes in the presence of one Byzantine fault.*

Proof. Assuming there is a protocol P that solves this problem, consider a system $C = C_0 \cup C_1 (C_0 \neq \phi, C_1 \neq \phi)$, in which C_0 is the set of nodes with input 0, and C_1 is the set of nodes with input 1. There exists a finite execution E of P in C that reaches an output-stable configuration in which the outputs of all nodes have stabilized to the same value. Without loss of generality assume the common output value is 0. Consider another system $C' = \{a\} \cup C_1$, in which a is a Byzantine node, and C_1 is the same as in C. Node a runs a two-phase protocol. In phase one, when it is a's turn to send a message, it nondeterministically chooses whether to remain in phase one or move to phase two. If it remains in phase one, it chooses one of the messages sent by nodes in C_0 in the execution E and sends that message to the recipient. Upon entering phase two, a faithfully imitates a nondeterministically chosen non-faulty node i from C_0 starting from the state i is in at the end of the execution E. There exists an execution E' of P in C' that simulates E, in the sense that every time there is a message in E sent between a node in C_0 and a node in C_1, there is a corresponding message in E' sent between a and the node in C_1, and at the end of E', node a will faithfully simulate one node in C_0. Thus, the configuration of the system C' at the end of E' is a subconfiguration of the system C at the end of E, and will continue so at every subsequent time. Thus the outputs of the non-faulty nodes (those in C_1) will remain 0 no matter how execution proceeds from this point. This violates the validity condition, because the inputs of all non-faulty nodes in C' are 1. □

The proof does not depend on the specific communication model and fairness assumption; therefore stabilizing Byzantine consensus is impossible even with the strong fairness condition and two-way interaction model in [1], and unbounded memory. Note that theorem 14 rules out not only deterministic solutions, but also randomized solutions[1], in the sense that for any candidate protocol P, there exists an $\epsilon_P > 0$, such that the probability of an execution failing to reach consensus is always greater than ϵ_P. ϵ_P is any constant less than the probability that C' successfully simulates C to the point when all non-faulty nodes are output-stable.

6 Discussion

6.1 Upper bound on faults

It was shown that in synchronous systems the number of Byzantine nodes must be strictly less than one third of the total number of nodes for any solution to the agreement problem [19, 25]. This bound still holds for stabilizing consensus in our model. We omit the proof here, because the original proof in [19] does not rely on synchrony and can be adapted to our model easily.

6.2 General graphs

In our model there is no limitation on the movement of nodes, except for the fairness condition. If we define an edge between every pair of nodes that can communicate with each other infinitely often, the graph is complete. In some applications nodes are restricted to a certain region, therefore the communication graph is not complete. It was proven in [11] that the Byzantine agreement problem can be solved in an n-node synchronous network graph G, tolerating f faults, if and only if the $n > 3f$ bound holds and G is at least $(2f + 1)$-connected. This result can also be transferred to our model. Intuitively, since G is at least $2f + 1$-connected, there are at least $2f + 1$ disjoint paths between any two nodes. Let each node send each message through $2f + 1$ disjoint paths. Then the majority of the copies the recipient receives are sent via paths that do not contain faults. Thus, it is possible to implement reliable communication between any two nodes, and the above algorithms still work for such communication graphs with messages sent over multi-hop links.

In some systems, nodes are moving around, but the fairness condition does not hold, meaning not every pair of nodes have infinitely many chances to communicate. Some pair of nodes can

[1] A randomized solution would guarantee that consensus be reached with probability 1, assuming some probabilistic distribution of the nodes' coin flips and the choices of the scheduler.

only send finitely many messages to each other, and some pairs won't get close enough to communicate at all. Such systems have an incomplete and changing communication graph. In their self-stabilizing group membership protocol, Dolev, Schiller and Welch [16] used random walks of a mobile agent as a means of information dissemination. Similarly, one or more non-Byzantine message carriers could be used as a link-layer service to implement end-to-end message passing between all pairs of nodes, thus simulating a complete communication graph. The message carriers do not have to be reliable as long as they successfully deliver messages infinitely often.

7 Conclusions and future work

In this paper we defined and investigated fault-tolerant stabilizing consensus in a model inspired by natural phenomena. We considered crash faults and Byzantine faults in fully asynchronous and decentralized mobile networks, as well as systems with stabilizing inputs and systems with incomplete or evolving connectivity. The algorithms are useful in controlling distributed systems, such as sensor networks, that simulate certain biological behaviors. They are also useful as a middleware layer that provides service to higher-level protocols. One drawback of the algorithm for stabilizing inputs is that it involves unbounded counters, unless there is a bound on the maximum number of times the inputs could change. It is open whether there exists a protocol for this problem with bounded memory. In many practical ad hoc networks, the graph representing possible communications changes over time. It is open for future research whether stabilizing consensus can be solved in these systems without additional message carriers, possibly using authentication and a fault-tolerant ad-hoc routing protocol.

References

[1] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. In *Ninth International Conference on Principles of Distributed Systems*, pages 79–90, December 2005.

[2] D. Barbara. Mobile computing and databases - a survey. *Knowledge and Data Engineering*, 11(1):108–117, 1999.

[3] Claudio Basile, Marc-Oliver Killijian, and David Powell. A survey of dependability issues in mobile wireless networks. Technical report, Laboratory for Analysis and Aarchitecture of Systems, National Center for Scientific Research, Toulouse, France, Feb 2003.

[4] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Synmposium on Principles of Distributed Computing*, pages 27–30, Montreal, Quebec, Canada, Aug 1983.

[5] Roberto Beraldi and Roberto Baldoni. *The handbook of ad hoc wireless networks*, pages 127–148. The Electrical Engineering Handbook Series. CRC Press, Inc. Boca Raton, FL, USA, 2003.

[6] C. Bobineau, P. Pucheral, and M. Abdallah. A unilateral commit protocol for mobile and disconnected computing. In *12th International Conference on Parallel and Distributed Computing Systems*, 2000.

[7] L. Briesemeister. *Group Membership and Communication inHIghly Mobile Ad Hoc Networks*. PhD thesis, School of Electrical Engineering and Computer Science, Technical University of Berlin, Germany, 2001.

[8] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, Jul 1996.

[9] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar 1996.

[10] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, Jul 1993.

[11] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, Mar 1982.

[12] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[13] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, Jul 1986.

[14] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the 14th annual ACM symposium on Theory of computing*, pages 401–407, San Francisco, California, United States, 1982.

[15] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal of Computing*, 12(4):656–666, Nov 1983.

[16] S. Dolev, E. Schiller, and J. Welch. Random walk for self-stabilizing group communication in ad-hoc networks. In *21st Symposium on Reliable Distributed Systems*, 2002.

[17] Paul Neil Feldman. *Optimal Algorithms for Byzantine Agreement*. PhD thesis, Massachusetts Institute of Technology, Jun 1988.

[18] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). Technical Report YALEU/DCS/TR-273, Yale University, 1983.

[19] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, Jan 1986.

[20] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, apr 1985.

[21] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 2002.

[22] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.)*. IEEEComputer Society Press, 1995.

[23] Leslie Lamport. The part-time parliament. *ACM Transaction on Computer Systems*, 16(2):133–169, May 1998.

[24] N. Malpani, J. L. Welch, and N. H. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proc. Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 96–103, 2000.

[25] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228–234, 1980.

[26] E. Pitoura and B. K. Bhargava. Data consistency in intermittently connected distributed systems. *Knowledge and Data Engineering*, 11(6):896–915, 1999.

[27] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science*, pages 403–409. IEEE, Los Alamitos, California, United States, 1983.

[28] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, pages 46–55, 1999.

[29] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet. Novel Type of Phase Transition in a System of Self-Driven Particles. *Physical Review Letters*, 75:1226–1229, August 1995.

[30] Jennifer E. Walter, Jennifer L. Welch, and Nitin H. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6):585–600, 2001.

[31] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 194–205, 2002.