Crash-Tolerant Consensus in Directed Graphs*

Lewis Tseng[1] and Nitin Vaidya[2]

[1] Department of Computer Science,
[2] Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign

Email: {ltseng3, nhv}@illinois.edu
Phone: +1 217-244-6024, +1 217-265-5414
Mailing address: Coordinated Science Lab., 1308 West Main St., Urbana, IL 61801, U.S.A.

December 30, 2014†

Abstract

This work considers a point-to-point network of n nodes connected by *directed* links, and proves *tight* necessary and sufficient conditions on the underlying communication graphs for achieving consensus among these nodes under *crash* faults. We identify the conditions in both synchronous and asynchronous systems.

*This research is supported in part by National Science Foundation awards 1329681. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

†Revised January 1, 2014 to make minor improvements to the presentation.

1 Introduction

In this work, we explore algorithms for achieving consensus in the presence of crash faults [9, 1]. We assume a point-to-point network, which is modeled as a *directed* graph, i.e., the communication links between neighboring nodes are not necessarily bi-directional. We consider both synchronous and asynchronous systems.

The crash consensus problem [9, 1] considers n nodes, of which at most f nodes may crash. The faulty nodes may fail stop at any point of time. We do not assume Byzantine behavior [8] in this work. A crash consensus algorithm is *correct* if it satisfies the following three properties:

- **Agreement**: the output (i.e., decision) at all the fault-free nodes is identical.
- **Validity**: the output at any fault-free node must be some node's input.
- **Termination**: every fault-free node eventually decides on an output.

This paper presents *tight* necessary and sufficient conditions for crash consensus in *directed* graphs.

1.1 Related Work

Lamport, Shostak, and Pease introduced the Byzantine consensus problem in [10]. Subsequently, researchers also explored the consensus problem in the presence of crash faults [1, 9]. It has been shown that the lower bound on the round complexity is $f + 1$, and $f + 1$ nodes are sufficient for solving crash consensus [1, 9]. For undirected graphs, it is easy to see that $f + 1$ node connectivity is both necessary and sufficient for crash consensus.

For Byzantine consensus in undirected graphs, [6, 4] showed that $2f + 1$ node connectivity is both necessary and sufficient. Recently, we identified tight conditions for Byzantine consensus problem in *directed* graphs [12]. For link failures in complete graphs, Schmid, Weiss, and Keidar proved impossibility results and lower bound on the number of nodes for synchronous consensus under *transient Byzantine link* faults [11]; however, the nodes are always fault-free. Many effort has also been devoted to characterizing tight conditions for other related problems. Please refer to our prior work [12] for more details.

For crash faults, Charron-Bost et al. proved tight conditions for *approximate* consensus in dynamic graphs [2], where the graphs may change continually and unpredictably, in synchronous and partially-synchronous systems. Our work considers *exact* and *approximate* consensus in synchronous and asynchronous systems, respectively. Moreover, we assume the communication graph is *static*.

1.2 Network Model

Sections 2 and 3 assume synchronous systems, and section 4 considers asynchronous systems. The underlying communication network is *static*, i.e., it does not change over time. The communication network consisting of n nodes is modeled as a simple *directed* graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes, and $\mathcal{E}$ is the set of directed edges between the nodes in $\mathcal{V}$. We assume that $n \geq 2$, since the consensus problem for $n = 1$ is trivial. Node i can transmit messages to another node j if and only if the directed edge (i, j) is in $\mathcal{E}$. Also, each node can send messages to itself as well. For node i, let N_i^-

be the set of nodes from which i can receive messages. That is, $N_i^- = \{\, j \mid (j,i) \in \mathcal{E}\,\} \cup \{i\}$. Define N_i^+ as the set of nodes that can receive messages from node i.That is, $N_i^+ = \{\, j \mid (i,j) \in \mathcal{E}\,\} \cup \{i\}$.

All the communication links are reliable, FIFO (first-in first-out) and deliver each transmitted message exactly once.

2 Synchronous Systems

2.1 Necessary Condition

All the paths we discuss in the paper are directed paths. We first introduce some useful definitions. A reduced graph G_F for a graph $G(\mathcal{V}, \mathcal{E})$ is a subgraph induced by vertex subset $\mathcal{V} - F$ where F is a potential fault set. The formal definition is presented below.

Definition 1 (Reduced Graphs) *For a given graph $G(\mathcal{V}, \mathcal{E})$, and a given parameter k, and each set $F \subset V$ such that $|F| \leq k$, reduced graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ is defined as follows: (i) $\mathcal{V}_F = \mathcal{V} - F$, and (ii) $\mathcal{E}_F$ is obtained by removing from $\mathcal{E}$ all the links incident on the nodes in F. That is, $\mathcal{E}_F = \mathcal{E} - \{(i,j) \in \mathcal{E} \mid i \in F or \;\; j \in F\}$.*

We define a fault-tolerant version of node connectivity over a directed graph, which extends the traditional notion of node connectivity (or vertex connectivity) [14].

Definition 2 (Crash-Tolerant Node Connectivity) *A graph $G(\mathcal{V}, \mathcal{E})$ is said to satisfy k Crash-Tolerant Node Connectivity (CT node connectivity) if for any $F \subset \mathcal{V}$ such that $|F| \leq k$, there is a single node $s \in \mathcal{V} - F$ that has paths to all the nodes in G_F.*

Recall that by assumption, we assume that $i \in N_i^+$ and N_i^-, and hence, i has a path to itself as well. The traditional notion of node connectivity [14], some reduced graph may not have a node that can reach all the nodes, since it only requires the reduced graph to be weakly connected.

Definition 3 (Directed Rooted Spanning Tree) *A spanning tree of a graph $H(\mathcal{V}, \mathcal{E})$ is said to be a directed rooted spanning tree if there is a single root in the spanning tree that has directed paths to all the nodes in $\mathcal{V}$.*

It should be easy to see that k CT node connectivity is equivalent to the condition that *given any reduced graph G_F, there exists a directed rooted spanning tree*. Charron-bost et al. [2] also use the notion of rooted spanning tree to specify the tight condition for achieving approximate consensus in dynamic networks.

Definition 4 (Source) *Given a reduced graph G_F, a node s is said to be the* source *of G_F if there exists a directed rooted spanning tree with s being the root.*

With a slight abuse of terminology, we will use the terms root and source interchangeably.

Based on CT node connectivity, the following theorem presents the necessary condition.

Theorem 1 *If exact consensus is possible in $G(\mathcal{V}, \mathcal{E})$ with at most f crash faults, then $G(\mathcal{V}, \mathcal{E})$ satisfies f CT node connectivity.*

Proof: The proof is by contradiction. Suppose that there exists a consensus algorithm, and $G(\mathcal{V}, \mathcal{E})$ does not satisfy f CT node connectivity. Thus, there exists a set $F \subset \mathcal{V}$ with $|F| \leq f$, and a pair of nodes $i, j \notin F$ such that there is no node s that has paths from s to both i and j. Note that by assumption, each node i has a path to itself.

For the reduced graph G_F and a node x in $\mathcal{V} - F$, define S_x as the set of all nodes that have paths to node x in G_F. Note that S_x contains x as well, because x has a path to itself. By assumption, S_i and S_j are disjoint. Moreover, there is no path from any node in S_i to any node in S_j in G_F, and vice versa, since otherwise, there exists some node that can reach both nodes i and j, which contradicts with the assumption. Then, V can be partitioned into disjoint sets F, S_i, S_j, R, where F, S_i and S_j are defined as above, and R contains the remaining nodes, i.e., $R = \mathcal{V} - F - S_i - S_j$. Then, we make the following observations:

- F and R may be empty, but S_i and S_j are non-empty, since $i \in S_i$ and $j \in S_j$.
- Nodes in R (if non-empty) have no path to nodes in $S_i \cup S_j$ in G_F by definition. This is because if some node $r \in R$ can reach some node in S_i or S_j in G_F, then by definition, r should also be in S_i or S_j, respectively. This contradicts with the assumption.

Now, consider an execution of the consensus algorithm where F (if non-empty) are the faulty nodes which crash before the start of the algorithm. All the other nodes are assumed to be fault-free. This is possible, since by assumption, $|F| \leq f$. Also, suppose that nodes in S_i and nodes in S_j have distinct input values. Without loss of generality, assume that nodes in S_i have input 0 and nodes in S_j have input 1. Nodes in R have input either 0 or 1.

Consider a node x in S_i. Since in G_F, there is no path from $S_j \cup R$ to nodes in S_i, the only input value learned by x throughout the execution of the algorithm is 0, and to satisfy validity property, 0 should be the output of x. Similarly, a node y in S_j can only learn 1 throughout the execution of the algorithm, and thus, 1 should also be the output of y. Note that by assumption, both S_i and S_j are non-empty, and fault-free. Therefore, the fact that S_i and S_j agree on different outputs violates the agreement property of the algorithm, a contradiction. □

2.2 Sufficiency

In this section, we propose a consensus algorithm in graphs that satisfy f CT node connectivity. This section assumes that each node has a binary input. In Section 2.3, we discuss how to extend the algorithm to solve multi-valued consensus. Note that the existence of such correct consensus algorithm proves the following theorem.

Theorem 2 *If $G(\mathcal{V}, \mathcal{E})$ satisfies f CT node connectivity, then binary consensus is achievable in G with at most f crash faults.*

This theorem also implies that f CT node connectivity is a *tight* condition for binary consensus. Section 2.3 shows that f CT node connectivity is sufficient for multi-valued consensus, as well. Therefore, f CT node connectivity is a *tight* condition for consensus in the presence of f crash faults.

Algorithm Min-Max For a graph H that contains a directed rooted spanning tree, define *height*(r, H) as the minimum height of all the spanning trees rooted at r in H. That is,

$$height(r, H) = \min_{\text{all spanning tree } T \text{ rooted at } r \text{ in } H} \text{height of } T$$

Given a graph G, define the fault-tolerant diameter d as follows:

$$d := \max_{F \subset \mathcal{V},\ |F| \leq f} \ \max_{\text{all roots } s \text{ of } G_F} height(s, G_F) \tag{1}$$

Due to the notion of directed rooted spanning tree (Definition 3), given any reduced graph G_F, if no node in $\mathcal{V} - F$ crashes, then the source of G_F (Definition 4) is able to propagate a value to any other node in $\mathcal{V} - F$ within d rounds by performing flooding, i.e., a source broadcasts its value in the first round, and then in later rounds, all the nodes forward the value received in the current round.

Now, we present the code running at each node i.

Algorithm Min-Max

```
• Set $v_i$ to the input at node $i$.
• For Phase $p = 1$ to $2f + 2$:
    If $p \ mod \ 2 = 0$, then repeat the following steps $d$ times (**Min Phase**):
        1. Broadcast $v_i$ to nodes in $N_i^+$.
        2. Receive the broadcast values from $N_i^-$.
        3. Set $v_i$ to the **minimum** value of all the values received.
    Else, repeat the following steps $d$ times (**Max Phase**:):
        1. Broadcast $v_i$ to nodes in $N_i^+$.
        2. Receive the broadcast values from $N_i^-$.
        3. Set $v_i$ to the **maximum** value of all the values received.
• Output $v_i$.
```

Note that by definition of, $i \in N_i^-$ and N_i^+, so in step 2 of each phase, i can receive the message from itself.

Theorem 3 *Algorithm Min-Max is correct for binary inputs in all the graphs that satisfy f CT node connectivity.*

Proof: Validity and termination properties are obvious, since d is upper bounded by n. Now, we prove that the agreement property also holds assuming that the inputs are either 0 or 1.

Fix an execution of the algorithm. Since there are $2f + 2$ phases. There must exists a pair of consecutive phases p_t, p_{t+1} such that no node crashes in phases p_t and p_{t+1}. Without loss of generality, let p_t be the Min Phase and p_{t+1} be the Max Phase.

Denote by F the nodes that have crashed in the execution by the end of Phase p_{t-1}. Recall that the source of a reduced graph H is defined as the root of the directed spanning tree in H as per Definition 4. Consider two cases:

- Case I: if some source s of the reduced graph G_F has $v_s = 0$ at the beginning of phase p_t, then due to the definition of the source and fault-tolerant diameter d, by the end of phase p_t, every node $i \in \mathcal{V} - F$ has received 0 on a path from source s and sets $v_i = 0$, since p_t is a Min Phase.

- Case II: if each source s of the reduced graph G_F has $v_s = 1$ at the beginning of phase p_t, then by the end of p_t, each source s still has $v_s = 1$. Suppose by way of contradiction that each source s of G_F has $v_s = 1$ at the beginning of phase p_t, but by the end of p_t, some source s' has $v_{s'} = 0$. By assumption, source s' must receive 0 on a path from some other *non-source* node x in phase p_t. However, the fact that there exists a path from x to s' implies that x is also a source in G_F, and $v_x = 0$ at the start of phase p_t. This is a contradiction. Now, observe that by the end of phase p_t, each source s still has $v_s = 1$, and phase p_{t+1} is the Max Phase. Therefore, by the end of p_{t+1}, every node $i \in \mathcal{V} - F$ will receive 1 on a path from source s and sets $v_i = 1$.

In either case, agreement is achieved. This completes the proof. □

2.3 Multi-valued Consensus

It is easy to see that Algorithm Min-Max does not work correctly when the input is not binary, since the source(s) of some reduced graph may not have either maximum or minimum input value(s), and thus, the rest of the nodes cannot learn the value(s) of the source(s) in either Min or Max Phase. This section considers the consensus problem with input being in the range $[0, K]$, where $K \geq 1$.

We present Algorithm MVC (Multi-Valued Consensus). It consists of two loops: The OUTER-LOOP iterates over all possible inputs, and the INNER-LOOP is essentially Algorithm Min-Max with an extra step to update the tentative state. In Algorithm MVC, each node i keeps track of two types of variables:

- t_i: This variable is the tentative state at each node. It is guaranteed that at any point of time, t_i equals an input at some node. Moreover, if node i enters OUTER-LOOP iteration l, then t_i is set to be some input value that has been seen by node i and is at least l.

- v_i: This *binary* variable acts as several roles. It first represents whether or not $t_i = l$ at the beginning of each OUTER-LOOP iteration l (STEP I of the OUTER-LOOP). Then, at the end of STEP II of the OUTER-LOOP, v_i becomes the output of Algorithm Min-Max (INNER-LOOP). Thus, at the beginning of STEP III of the OUTER-LOOP, nodes will have the same v_i's, which allows nodes to reach an agreement on whether to proceed to next OUTER-LOOP iteration.

Now, we describe the structure of Algorithm MVC. In each OUTER-LOOP iteration $l \in [0, K]$, nodes try to learn whether some node i has the tentative state $t_i = l$ at the beginning of the iteration. First, v_i acts as a local observation at node i, i.e., v_i is set to 0 if and only if $t_i = l$ (STEP I of the OUTER-LOOP). Then, at STEP II of the OUTER-LOOP, nodes use Algorithm Min-Max (INNER-LOOP) to reach agreement on the observations (v_i's). There are two possible outcomes at the end of the STEP II of the OUTER-LOOP:

- $v_i = 0$:

This case implies that nodes learn that some node i has $t_i = l$ at the beginning of the OUTER-LOOP iteration, and they know that all the other nodes that have not crashed also learn the same information. Thus, nodes will exit the OUTER-LOOP with outputs l (STEP III of the OUTER-LOOP).

- $v_i = 1$:

  In this case, nodes will proceed to the next OUTER-LOOP iteration.[1] Moreover, nodes are guaranteed to set their tentative state (t_i's) to some value strictly greater than l when completing the INNER-LOOP. At step 4 of each INNER-LOOP phase, nodes update t_i's to the minimum value that is strictly greater than l and is received in that INNER-LOOP phase. Later, we will show that if at any point of time, node i changes v_i from 0 to 1, then t_i will also be updated to some value strictly greater than l. Thus, if nodes enter the OUTER-LOOP iteration $l+1$, then no node will ever have tentative state $\leq l$. If at the end of OUTER-LOOP K, nodes do not exit the loop, i.e., the code *Exit OUTER-LOOP* is never executed, then all the fault-free nodes will terminate with output K.

Note that due to the agreement property of Algorithm Min-Max, either nodes will exit OUTER-LOOP at the same iteration, or nodes will terminate with output K.

Algorithm MVC

```
• $t_i[0]$ := input at node $i$
• OUTER-LOOP $l := 0$ to $K$:
    – STEP I: If $t_i[l] == l$, then $v_i[l] := 0$; otherwise, $v_i[l] := 1$
    – STEP II: INNER-LOOP $p := 1$ to $2f + 2$:
        Repeat the following steps $d$ times:
        1. Broadcast the tuple $(v_i[l], t_i[l])$
        2. Receive the broadcast tuples from incoming neighbors and node $i$ itself. Denote by
           $B_i$ the set of tuples received in this step.
        3. If $p$ mod $2 = 0$, then                                  \\ Min-Phase
               $v_i[l] := \min\{a \mid (a, *) \in B_i\}$
           Else,                                                      \\ Max-Phase
               $v_i[l] := \max\{a \mid (a, *) \in B_i\}$
        4. If $|\min\{b \mid (*, b) \in B_i,\ b > l\}| > 0$, then
               $t_i[l] := \min\{b \mid (*, b) \in B_i,\ b > l\}$
    – STEP III: If $v_i[l] == 0$, then
          Exit OUTER-LOOP
• Output $l$
```

Theorem 4 *Algorithm MVC is correct in all the graphs that satisfy f CT node connectivity.*

The proof is presented in Appendix A.

[1] Note that this case does not mean that *no* node has $t_i = l$. However, in this case, nodes cannot be sure that all nodes that have not crashed also have learned that some node i has $t_i = l$ at the beginning of the OUTER-LOOP iteration. Thus, nodes have to proceed to the next OUTER-LOOP iteration.

3 Iterative Algorithms

Observe that Algorithm Min-Max does not utilize any topology information, since it does not require node identifiers (ID), and the usage of the fault-tolerant diameter d can be replaced by the number of nodes n. That is, assuming the knowledge of n and f, Algorithm Min-Max works in *anonymous* systems [7] and *anonymous* networks [3], where nodes do not have IDs. In anonymous systems, we define a family of iterative algorithms – *Fixed Iterative Algorithm* – those iterative algorithms using *fixed* transition functions. This section assumes synchronous systems, as well.

Iterative Algorithms We first describe the structure of the iterative algorithms of interest. Each node i maintains state v_i, with $v_i[t]$ denoting the state of node i at the *end* of the t-th iteration of the algorithm. Initial state of node i, $v_i[0]$, is equal to the initial *input* provided to node i. At the *start* of the t-th iteration ($t > 0$), the state of node i is $v_i[t-1]$. The iterative algorithms of interest will require each node i to perform the following three steps in iteration t, where $t > 0$.

1. *Transmit step:* Transmit current state, namely $v_i[t-1]$, on all outgoing edges.
2. *Receive step:* Receive values on all incoming edges. Denote by $r_i[t]$ the union of i's value and the values received by node i from its neighbors.
3. *Update step:* Node i updates its state using a transition function Z_i as follows. Z_i is a part of the specification of the algorithm, and takes as input the vector $r_i[t]$.

$$v_i[t] \quad = \quad Z_i \ (r_i[t], t) \qquad (2)$$

Fixed Iterative Algorithms

Definition 5 (Fixed Transition Function) *A transition function Z_i for node i is said to be* fixed *if for all iteration $t \geq 0$ and all $i \in \mathcal{V}$, $Z_i(R_i[t], t) = Z^*(R_i[t])$. In other words, the transition function does not change over time, and every node uses the same transition function.*

For iterative algorithms that use fixed transition function, we present the following result.

Theorem 5 *In general, it is impossible to solve consensus using* fixed iterative algorithms *in anonymous systems and networks.*

The proof is presented in Appendix B.

4 Asynchronous Systems

This section considers asynchronous systems, where each node proceeds in different speed and the messages may be arbitrarily delayed. For simplicity, we assume the channels are reliable.

Approximate Consensus [6] showed that it is impossible to achieve exact consensus in asynchronous systems with a single crash fault. Therefore, we are interested in approximate consensus algorithms. The algorithms must achieve the following three properties:

- **ϵ-agreement**: the difference between outputs at any two fault-free nodes is bounded by ϵ.

- **Validity**: the output at any fault-free node is in the *convex hull* of all the inputs.
- **Termination**: every fault-free node decides on an output in a finite-amount of time.

4.1 Necessity

To facilitate the discussion, we first introduce an useful definition.

Definition 6 *Given a graph $G(\mathcal{V},\mathcal{E})$ and a node-partition A, B of $\mathcal{V}$, A is said to* **propagate** *to B if (i) B is not empty; and (ii) there exist at least $f+1$ distinct nodes in A which have outgoing links to some node in B, i.e., $|\{i \mid i \in A, \quad N_i^+ \cap B \neq \emptyset\}| \geq f+1$.*

We will denote the fact that set A propagates to set B by the notation of $A \rightarrow B$. When it is not true that $A \rightarrow B$, we will denote that fact by $A \not\rightarrow B$.

Theorem 6 *Suppose that an asynchronous approximate consensus algorithm exists for $G(\mathcal{V},\mathcal{E})$. Then for any node partition L, C, R of $\mathcal{V}$, where L and R are both non-empty, either $L \cup C \rightarrow R$ or $C \cup R \rightarrow L$.*

Proof: The proof is by contradiction. Suppose that there exists a correct approximate consensus algorithm, and $G(\mathcal{V},\mathcal{E})$ does not satisfy the condition. That is, there exists a node partition L, C, R such that L and R are not empty, and $L \cup C \not\rightarrow R$ and $C \cup R \not\rightarrow L$. Let $O(L)$ denote the set of nodes in $C \cup R$ that have outgoing links to some nodes in L, i.e., $\{i \mid i \in C \cup R, \quad N_i^+ \cap L \neq \emptyset\}$. Similarly, define $O(R) = \{j \mid j \in L \cup C, \quad N_j^+ \cap R \neq \emptyset\}$. By assumption, $|O(L)| \leq f$ and $|O(R)| \leq f$.

Consider the scenario where (i) each node in L has input 0; (ii) each node in R has input 2ϵ; (iii) nodes in C (if non-empty) have arbitrary inputs in $[0, 2\epsilon]$; (iv) no node crashes; and (v) the message delay from $O(L)$ to L and from $O(R)$ to R is arbitrarily large compared to all the other traffic. Consider nodes in L. From their perspectives, it is possible that all nodes in $O(L)$ have crashed. This is due to the following observations:

- The only nodes in $C \cup R$ that have outgoing links to L are nodes in $O(L)$. Thus, nodes in L are not able to learn whether nodes in $O(L)$ are alive or not from nodes in $(C \cup R) - O(L)$.
- The message delay from $O(L)$ is arbitrarily large.
- The size of $|O(L)| \leq f$.

Therefore, nodes in L cannot wait for any message from nodes in $O(L)$ to decide the outputs. Similarly, nodes in R cannot wait for any message from nodes in $O(R)$ to decide the outputs. Consequently, to satisfy the validity property, the output at each node in L has to be 0, since 0 is the input of all the nodes in L. Similarly, all nodes in R have to output 2ϵ. Thus, ϵ-agreement property is violated, since $\epsilon < 2\epsilon$. This is a contradiction. □

4.2 Sufficiency

We prove that the condition in Theorem 6 is also sufficient by proposing an asynchronous approximate consensus algorithm – Algorithm WA (Wait-and-Average). The algorithm assumes the knowledge of global topology at each node, and the algorithm proceeds in phases. In each phase, nodes flood messages containing their current value, ID (identifier), and a phase index. Each node i waits until it has received *enough* values from other nodes. Then, node i updates its value to be the *average* of all the values received in this phase, and then proceeds to the next phase. When node i has finished p_{end} phases, it outputs its current state. p_{end} is some sufficiently large integer.

Now, we discuss how many values received by a node is considered *enough*. Let $heard_i[p]$ be the set of nodes from which node i has received values *during* phase p. Each node i proceeds to perform the averaging operation if the following condition holds.

Condition *WAIT*: Denote by $reach_i(F)$ the set of nodes that have paths to node i in the reduced graph G_F. Then, Condition *WAIT* is satisfied if there exists a set of nodes $F_i \subseteq \mathcal{V} - \{i\}$ and $|F_i| \leq f$ such that $reach_i(F_i) \subseteq heard_i[p]$.[2]

Now, we present the algorithm below.

Algorithm WA

```
$p_{end}$ is some sufficiently large integer.

• For each node $i$, set $v_i[0]$ to the input at node $i$
• For Phase $p = 1$ to $p_{end}$:
    – On entering phase $p \geq 1$:
        $R_i[p] = \{v_i[p-1]\}$
        $heard_i[p] = \{i\}$
        Send message $(v_i[p-1], i, p)$ to all the outgoing neighbors

    – When message $(h, j, p)$ is received for the first time:
        $R_i[p] = R_i[p] \cup \{h\}$
        $heard_i[p] = heard_i[p] \cup \{j\}$
        Send message $(h, j, p)$ to all the outgoing neighbors
        if Condition WAIT holds:
            $v_i[p] = \frac{\sum_{v \in R_i[p]} v}{|R_i[p]|}$
• Output $v_i$
```

The following theorem shows the correctness of Algorithm WA. It also proves that the condition in Theorem 6 is sufficient for approximate consensus in asynchronous systems.

Theorem 7 *Algorithm WA is correct in all graphs that satisfy the condition in Theorem 6.*

[2] $reach_i(F_i)$ may be different in each phase, since it depends on the delay pattern. For simplicity, we ignore the phase index p in the notation.

Proof Sketch: Validity and termination properties are obvious. For ϵ-agreement, we only present the key lemma here. The rest of the proof is standard, e.g., [13, 5, 1].

For phase $p \geq 1$, consider two nodes i, j that have successfully computed values $v_i[p]$ and $v_j[p]$, respectively, in phase p. That is, i and j have not crashed before computing $v[p]$'s. With a slight abuse of terminology, define $heard_i[p]$ as the set of nodes whose values are used by node i to compute its state $v_i[p]$ in phase p. Define $heard_j[p]$ similarly.

Lemma 1 *$heard_i[p] \cap heard_j[p] \neq \emptyset$.*

Proof: By construction, there exist two sets F_i and F_j such that (i) $F_i \subseteq \mathcal{V} - \{i\}$ and $|F_i| \leq f$; (ii) $F_j \subseteq \mathcal{V} - \{j\}$ and $|F_j| \leq f$; (iii) $reach_i(F_i) \subseteq heard_i[p]$; and (iv) $reach_j(F_j) \subseteq heard_j[p]$. If $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, then the proof is complete, since $reach_i(F_i) \subseteq heard_i[p]$ and $reach_j(F_j) \subseteq heard_j[p]$. Thus, $heard_i[p] \cap heard_j[p] \neq \emptyset$. Now, consider the case when $reach_i(F_i) \cap reach_j(F_j) = \emptyset$. We will derive a contradiction in this case.

We start with the following claim:

Claim 1 *In G, the only nodes that may have outgoing links to nodes in $reach_i(F_i)$ are nodes in F_i. Similarly, in G, the only nodes that may have outgoing links to nodes in $reach_j(F_j)$ are nodes in F_j.*

Proof: Recall that $reach_i(F_i)$ is defined as the set of nodes that have paths to node i in the reduced graph G_{F_i}, and $reach_j(F_j)$ is defined similarly. Thus, $F_i \cap reach_i(F_i) = \emptyset$ and $F_j \cap reach_j(F_j) = \emptyset$. These two observations together with the definitions of $reach_i(F_i)$ and $reach_j(F_j)$ imply that there is no path from nodes in $\mathcal{V} - reach_i(F_i) - F_i$ (if non-empty) to nodes in $reach_i(F_i)$ in G_{F_i}. Hence, the claim is proved. □

Let $L = reach_i(F_i)$, $R = reach_j(F_j)$ and $C = \mathcal{V} - L - R$. Observe that since $reach_i(F_i) \cap reach_j(F_j) = \emptyset$, L, C, R form a partition of $\mathcal{V}$. Moreover, $i \in reach_i(F_i)$ and $j \in reach_j(F_j)$; hence, $L = reach_i(F_i)$ and $R = reach_j(F_j)$ are both non-empty. Then, let $O(L)$ be the nodes in $C \cup R$ that have outgoing links to some nodes in L in G. Since $L = reach_i(F_i)$, the only nodes that may be in $O(L)$ are in F_i due to Claim 1. By assumption, $|F_i| \leq f$. Therefore, $C \cup R \not\to L$. Similarly, we can argue that $L \cup C \not\to R$. These two conditions violate the necessary condition, a contradiction. Thus, $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, which implies $heard_i[p] \cap heard_j[p] \neq \emptyset$. This completes the proof. □

Let M and m denote the upper bound and the lower bound on the inputs, respectively. Then, by an analysis similar to [13, 5, 1], Lemma 1 can be used to show ϵ-agreement when p_{end} is sufficiently large (as a function of n, f, M, m).

□

5 Summary

This paper addresses consensus problems in the presence of crash faults, where the underlying communication networks may be incomplete. We explore exact and approximate consensus algorithms in synchronous and asynchronous systems, respectively. We prove *tight* conditions for the graphs to be able to solve these consensus problems.

References

[1] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Series on Parallel and Distributed Computing, 2004.

[2] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks. *CoRR*, abs/1408.0620, 2014.

[3] C. Delporte-Gallet, H. Fauconnier, and A. Tielmann. Fault-tolerant consensus in unknown and anonymous networks. In *ICDCS*, pages 368–375. IEEE Computer Society, 2009.

[4] D. Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):1430, March 1982.

[5] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.

[6] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, PODC '85, pages 59–70, New York, NY, USA, 1985. ACM.

[7] R. Guerraoui and E. Ruppert. What can be implemented anonymously? In P. Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2005.

[8] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 1982.

[9] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[10] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, Apr. 1980.

[11] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM J. Comput.*, 38(5):1912–1951, Jan. 2009.

[12] L. Tseng and N. H. Vaidya. Exact byzantine consensus in directed graphs. *CoRR*, abs/1208.5075, 2012.

[13] L. Tseng and N. H. Vaidya. Asynchronous convex hull consensus in the presence of crash faults. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 396–405, New York, NY, USA, 2014. ACM.

[14] D. B. West. *Introduction To Graph Theory.* Prentice Hall, 2001.

A Proof of Theorem 4

Theorem 4 Algorithm MVC is correct in all the graphs that satisfy f CT node connectivity.

Proof: The termination property is obvious. Now, we prove the two other properties. Suppose that the graph $G(\mathcal{V}, \mathcal{E})$ satisfies the condition stated in Theorem 1. Let $v_i^{end}[l]$ be the value $v_i[l]$ at node i after the INNER-LOOP is completed in some OUTER-LOOP iteration l.

Claim 2 *For all nodes i, j that have not crashed in OUTER-LOOP iteration l, $v_i^{end}[l] = v_j^{end}[l]$.*

Proof: This is due to the correctness of Algorithm Min-Max, since if we ignore the code related to t_i's, then the INNER-LOOP is essentially equal to Algorithm Min-Max. □

We will use Claim 2 to prove the agreement property. In the proof below, we will say that a node *exits* an OUTER-LOOP iteration l if it has $v_i^{end}[l] = 0$; otherwise, a node is said to *complete* the iteration l.

Lemma 2 *Algorithm MVC satisfies the agreement property in G.*

Proof: By Claim 2, all the nodes that have not crashed will either exit the OUTER-LOOP in the same iteration l or complete OUTER-LOOP iteration K. Thus, all the fault-free nodes will have the same output l. □

To prove the validity property, we first introduce some notations, and prove useful lemma and claims. Let $t_i^{begin}[l]$ be the value $t_i[l]$ at node i at the *beginning* of some OUTER-LOOP iteration l, and let $t_i^{end}[l]$ be the value $t_i[l]$ at node i at the *end* of OUTER-LOOP iteration l. Let $v_i^{begin}[l]$ be the value $v_i[l]$ at node i after STEP I of the OUTER-LOOP iteration l. Thus, if $t_i^{begin}[l] = l$, then $v_i^{begin}[l] = 0$; otherwise, $v_i^{begin}[l] = 1$.

Lemma 3 *In an OUTER-LOOP iteration l $(0 \leq l < K)$, for each node $i \in \mathcal{V}$ that has not crashed, and has $v_i[l] = 1$, then $t_i[l] > l$.*

Proof: The proof is by induction on OUTER-LOOP iterations.

Induction Basis: $l = 0$.

We first prove the following claim.

Claim 3 *At any point of time, for each node i that has not crashed, and has $v_i[0] = 1$, then $t_i[0] > 0$.*

Proof: First, we prove the following claim: each node i will change $v_i[0]$ from 0 to 1 if and only if it receives $(1, x)$ from its incoming neighbor such that $x > 0$. The proof is by contradiction. Consider the first Max-Phase p (of the INNER-LOOP) in which some node i changes $v_i[0]$ from 0 to 1, because i has received $(1, 0)$ from its incoming neighbors. Then, consider a chain of nodes propagating the tuple $(1, 0)$ from some node s to node i such that node s has $v_s[0] = 1$ and $t_s[0] = 0$ at the beginning of the Max-Phase p. Note that by assumption of p, node s has never received $(1, 0)$ from other nodes before Max-Phase p. Moreover, node s has also never received $(1, x)$ such that $x > 0$ from other nodes before Max-Phase p, since otherwise, $t_s[0]$ would be updated to x at step 4 of the INNER-LOOP. These two observations imply that $v_s[0] = 1$ and $t_s[0] = 0$ before entering the INNER-LOOP, i.e., after line 1 of the OUTER-LOOP is executed. This is a contradiction.

Second, Claim 3 follows directly from the claim above.

□

Claim 3 implies that the statement of Lemma 3 holds for the base case $(l = 0)$.

Induction Step: Suppose that for all OUTER-LOOP iteration $l \geq r$, the statement of Lemma 3 holds. Consider the $(r + 1)$-th OUTER-LOOP iteration. We can prove the following claim based on similar logic as in the base case and the induction hypothesis.

Claim 4 *At any point of time, for each node i that has not crashed and has $v_i[r+1] = 1$, then $t_i[r+1] > r+1$.*

This claim completes the proof of induction step. Thus, Lemma 3 is proved. □

Claim 5 *At any point of time in an OUTER-LOOP iteration l, if node i has not crashed, then $t_i[l]$ equals an input at some node.*

Proof: This claim holds by construction, since all the t's propagated are initially some node's input. □

Claim 6 *If any node i exits OUTER-LOOP iteration l and outputs l, then there must exist some node j such that $t_j^{begin}[l] = l$.*

Proof: Suppose by way of contradiction that every node j that has not crashed has $t_j^{begin}[l] \neq l$, and node i exits iteration l. The first assumption implies that every node j has $v_j^{begin}[l] = 1$. Due to the validity of Algorithm Min-Max, every node j that has not crashed after completing INNER-LOOP has $v_j^{end}[l] = 1$. Therefore, no node will exit iteration l, a contradiction. □

Now, we are ready to prove the key lemma.

Lemma 4 *Algorithm MVC satisfies the validity property in G.*

Proof: Consider two cases:

- Some node has input K:

  In this case, suppose that all the fault-free nodes exit the OUTER-LOOP iteration $l \leq K$ and output l. Then, by Claims 5 and 6, the validity property holds. Suppose that no fault-free node exits the OUTER-LOOP, i.e., for all i that has not crashed, $v_i^{end}[K] = 1$. In this case, the validity property still holds, since all the fault-free nodes will output K, and by assumption, some node has input K.

- No node has input K:

  Assume that all the nodes have input $\leq K'$, where $K' < K$. In this case, we show the following claim.

  Claim 7 *All the fault-free nodes will exit the OUTER-LOOP in some iteration $l \leq K'$.*

  Proof: If fault-free nodes exit during some OUTER-LOOP iteration $l < K'$, then the proof is done. Suppose not. Then, in iteration $K' - 1$, every node i that has not crashed has $v_i^{end}[K'-1] = 1$. Consequently, by Lemma 3, every node that has not crashed has $t_i^{end}[K'-1] > K'-1$. This observation together with Claim 5 and the assumption that the input is bounded by K' imply that $t_i^{end}[K'-1] = K'$. Therefore, in the beginning of iteration K', every node that has not crashed has $t_i^{begin}[K'] = K'$ and $v_i^{begin}[K'] = 0$. Then, due to the validity property of Algorithm Min-Max, every node i that has not crashed has $v_i^{end}[K'] = 0$. Therefore, every fault-free node will exit the OUTER-LOOP in iteration K'. □

  Claims 5, 6 and 7 together prove the validity property.

□

Lemmas 2 and 4 prove Theorem 4. □

B Proof of Theorem 5

Theorem 5 In general, it is impossible to solve consensus using *fixed iterative algorithms* in anonymous systems and networks.

Proof: We prove the theorem by showing a counter example. We present a directed graph that satisfies f CT node connectivity, and show that no fixed transition function solves consensus.

Consider a directed graph G consisting of three parts: (i) a clique of size $f+1$, (ii) a source node s that has an outgoing edge to every node in the clique, and (iii) a leaf node l that has an incoming edge from every node in the clique. Note that there is no incoming edge to s, and no outgoing edge from l. Moreover, edge (s, l) is not an edge in G. Obviously, the graph satisfies f CT node connectivity, since (i) if $s \in F$, then at least one node in the clique is the source in the reduced graph G_F; (ii) if $s \notin F$, then s is the source in G_F.

Suppose that each node uses the transition function Z. First, we look at how Z maps to a value when a node receives exactly $f+1$ values. Recall that $R_i[t]$ denotes the set of values received by i at iteration t. It is clear that if $R_i[t]$ contains all 0's or all 1's, then $Z(R_i[t])$ should map to 0 or 1, respectively; otherwise, either validity or agreement property is violated. This implies the following claim:

Claim 8 *There must exist a pair of set of $2f+1$ values R^0 and R^1 such that (i) $|R^0| = |R^1| = f+1$; (ii) there is exactly one more 1 in R^1 than in R^0, i.e., suppose R^0 contains a 0's and $(f+1-a)$ 1's, then R^1 contains $(a-1)$ 0's and $(f+2-a)$ 1's; and (iii) $Z(R^0) = 0$ and $Z(R^1) = 1$.*

Denote by R the set of f 0's and one 1, and R' the set of f 1's and one 0. Claim 8 implies that there are three possible cases. In all the cases below, we consider an execution of the algorithm where (i) a single node in the clique crashes before the algorithm starts; and (ii) no other node crashes throughout the execution. The inputs at each node is described in each case below.

- $Z(R) = 0$:

  Consider the case when the source s has input 1, and all the other nodes have input 0. Since the source node does not receive any value, its state can only be 1 throughout the execution. For each node in the clique in each iteration $t \geq 0$, it receives f 0's and one 1, and thus, state at each node in the clique can only be 0, since $Z(R) = 0$. Thus, the agreement property is violated.

- $Z(R) = 1$ and $Z(R') = 0$:

  Consider the case when the leaf l has input 0, and all the other nodes have input 1. Since the source node does not receive any value, its state can only be 1 throughout the execution. For each node in the clique in each iteration $t \geq 0$, it receives $f+1$ 1's, and thus, state at each node in the clique can only be 1. As a result, in each iteration t, the leaf node l receives f 1's and one 0, and thus the state at node l is 0 in each iteration, since $Z(R') = 0$. Thus, the agreement property is violated.

- $Z(R) = 1$ and $Z(R') = 1$:

  Consider the case when the source s has input 0, and all the other nodes have input 1. Since the source node does not receive any value, its state can only be 0 throughout the execution.

For each node in the clique in each iteration $t \geq 0$, it receives f 1's and one 0, and thus, state at each node in the clique can only be 1, since $Z(R') = 1$. Thus, the agreement property is violated.

□