Distributed Computing Column 63
A Note on Fault-tolerant Consensus in Directed Networks

Jennifer L. Welch
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843-3112, USA
welch@cse.tamu.edu

This column consists of an article by Lewis Tseng and Nitin Vaidya on fault-tolerant consensus in *directed* networks. The fault-tolerant consensus problem was initially studied for undirected complete graphs, and then for undirected arbitrary graphs. However, there was very little work on directed graphs, especially arbitrary ones, until recently. This article provides an overview of the authors' work on identifying exactly what conditions on the topology of directed graphs are necessary and sufficient for solving consensus (exact/approximate) in various models (crash/Byzantine failures, synchronous/asynchronous) with either iterative or general algorithms. The authors point out interesting connections to papers in the control theory literature and leave us with a couple of intriguing open questions

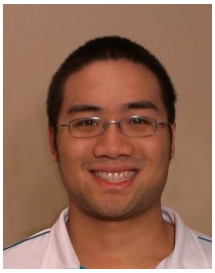
Many thanks to Lewis and Nitin for their contributions!

Call for contributions: I welcome suggestions for material to include in this column, including news, reviews, open problems, tutorials and surveys, either exposing the community to new and interesting topics, or providing new insight on well-studied topics by organizing them in new ways.

A Note on Fault-tolerant Consensus in Directed Networks

Lewis Tseng
Toyota InfoTechnology Center
USA
ltseng@us.toyota-itc.com

Nitin H. Vaidya
University of Illinois at Urbana-Champaign
USA
nhv@illinois.edu



1 Introduction

Fault-tolerant *consensus* has received significant attention over the past three decades since the seminal work by Lamport, Shostak, and Pease [33, 26]. The consensus problem considers n nodes, of which at most f nodes may be faulty. In this article, we address crash faults and Byzantine faults both. Each node is given an input value initially, and after a finite amount of time, each fault-free node should produce an output value, which satisfies appropriate *validity* and *agreement* conditions. This article explores the problem of using deterministic algorithms to achieve *exact* and *approximate* consensus in directed message-passing networks, which are modeled as *directed* graphs. We survey the results in both synchronous and asynchronous systems. Generally speaking, for exact crash-tolerant consensus, the output at the nodes must equal the input of one of the nodes. For approximate crash-tolerant consensus, the output must be in the range of the inputs of all the nodes. For exact Byzantine consensus, the validity condition depends on whether the inputs are binary or not: for binary inputs, the output must be the input of a fault-free node, whereas, for multi-valued inputs, the output must equal the input of fault-free nodes when they all have the same input. Finally, for approximate Byzantine consensus, the output must be in the range of the inputs of all the *fault-free* nodes.

Historically, fault-tolerant consensus in message-passing networks has been studied extensively both in complete graphs (e.g., [33, 15, 16, 1, 25]) and in undirected graphs (e.g., [17, 13]). The tight conditions on undirected graphs identified in these papers are summarized in Table 1. The term *connectivity* is used here to mean *node connectivity* [47] — removal of up to $\kappa - 1$ nodes does not cause a κ-connected graph to become partitioned. We will often use the terms *graph* and *network* interchangeably.

Recently, significant efforts [42, 37, 46, 11, 27, 28, 49, 29] have been devoted to solving the problem in *incomplete directed networks*, i.e., not every pair of nodes is connected by a communication channel, and the communication channels are not necessarily bi-directional. The goal of these works

Table 1: Tight Conditions on Undirected Networks

	Crash-Tolerant Consensus	**Byzantine Consensus**
Synchronous	$(f+1)$-connectivity, $n > f$ (follow from well-known results [30, 3])	$(2f+1)$-connectivity, $n > 3f$ ([17, 13])
Asynchronous	$(f+1)$-connectivity, $n > 2f$ (follow from well-known results [30, 3])	$(2f+1)$-connectivity, $n > 3f$ (follows from [1, 17])

is to fully characterize the *directed* graphs in which consensus is possible (under different system and fault models). We first identify why results in undirected networks (Table 1) do not directly apply to directed networks. Then, we present results in directed networks and briefly discuss the intuition.

1.1 Tight Conditions on Undirected Networks

Consider the exact crash-tolerant consensus problem in synchronous systems. From Table 1, we know that $(f+1)$-connectivity, and $n > f$ are together necessary and sufficient for achieving consensus [30, 3] in undirected network case. There are two straightforward ways to generalize the notion of connectivity in undirected networks to directed networks:

- $(f+1)$-*weak connectivity* [47]: removing up to f nodes, the remaining graph is weakly connected.

  A directed graph is weakly connected if in the graph, any node is reachable from any other node by traversing edges in some direction (not necessarily in the direction of the edge). It should be obvious to see that weak connectivity is necessary, since nodes need to exchange some information in order to achieve consensus. However, $(f+1)$-weak connectivity is not sufficient for crash-tolerant consensus. Consider the network in Figure 1. Consensus is impossible to achieve even if $f = 0$ (every node is fault-free), since nodes v_1 and v_2 cannot observe any common information.

- $(f+1)$-*strong connectivity* [47]: removing up to f nodes, the remaining graph is strongly connected.

  A directed graph is strongly connected if in the graph, any node is reachable from any other node by traversing edges following their directions. $(f+1)$-strong connectivity and $n > f$ are together sufficient, because we are able to route messages between any pair of fault-free nodes even if up to f nodes crash, which can be used to implement the reliable broadcast primitive among fault-free nodes; thus, simulation of any crash-tolerant consensus algorithm in complete graphs is possible in such directed graphs (assuming synchrony). To see that $(f+1)$-strong connectivity is not necessary, consider Figure 2. This network is *not* strongly connected; yet, when $f = 0$ (every node is fault-free), consensus is possible by having both nodes choose node v_1's input as output.

Asymmetric Information Flow These two generalizations are not adequate for specifying *tight* conditions on directed networks because they do not completely capture the notion of "asymmetric

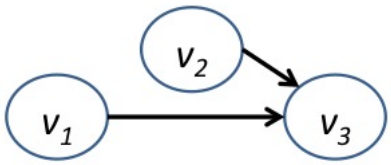


Figure 1: A weakly-connected network.

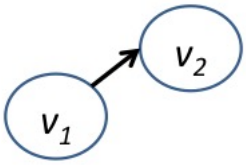


Figure 2: A network that is *not* strongly-connected.

information flow". One way to capture such notion is by the usage of a "reduced graph" [42, 37, 46] as shown in the following claim.

Claim 1. *It is possible to achieve crash-tolerant consensus in synchronous systems in a graph G while tolerating up to f crash faults if and only if after removing up to f nodes and all the links incident to the removed nodes in G, there exists a* source *node that can reach other nodes in the remaining graph (reduced graph), i.e., there exists a directed rooted spanning tree in the reduced graph.*

In the claim, the "remaining graph" is the reduced graph for crash-tolerant consensus in synchronous systems [42]. Due to the existence of a directed rooted spanning tree in each reduced graph, information disseminated by at least one *source* node (specified in the claim) can be shared by all the fault-free nodes. Later, we discuss other forms of reduced graphs for other consensus problems. Note that for a given G and a given f, there may exist multiple reduced graphs. The condition requires that a directed rooted spanning tree exists in all possible reduced graphs. In the fault-free case ($f = 0$), the condition requires the existence of a directed rooted spanning tree in G. For example, the network in Figure 2 satisfies the condition; whereas, the network in Figure 1 does not satisfy the condition. Now, consider the example network in Figure 3. This network satisfies the condition in Claim 1 for $f = 1$, since after removing up to 1 node, the reduced graph is a directed rooted spanning tree.

It should be straightforward to observe that the condition in Claim 1 is necessary, since if there is no such spanning tree after removing up to f nodes, then we can find a failure pattern to "block" the flow of any shared information between some pair of fault-free nodes. It is less obvious why the condition is sufficient, since the spanning tree may change over time because some nodes crash. Later, we discuss a consensus algorithm from [42] that relies on the observation that no matter how the failures occur, there is a directed rooted spanning tree that can "propagate" the information. In the algorithm, each node does not know the structure of the spanning tree; however, the algorithm still ensures that *at some point of time*, enough information can be propagated to all the nodes that have not crashed yet, and each node can use this information to achieve consensus.

Observations similar to Claim 1 were first made in the context of fault-free consensus [5, 22], and also in the context of various versions of fault-tolerant consensus problems [11, 42, 37, 46]. The exact manner in which the source node is identified differs for the different problems because of different assumptions on fault behaviors, system models and problem specifications.

1.2 Fault-tolerant Consensus Problem

We discuss the definition of the consensus problems here. In this article, we limit our consideration to *scalar* input and output. There are four dimensions to formally define a consensus problem —

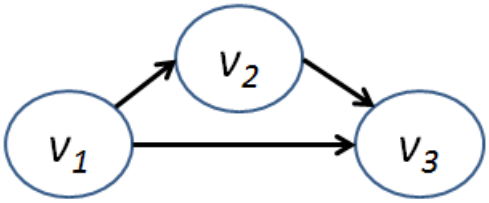


Figure 3: An example network that tolerates one crash fault.

output type, fault model, system synchrony assumption, and algorithm type. We briefly discuss each of the dimensions. The models are standard assumptions in the literature. Interested readers may refer to [30, 3] for more details.

- *Output Type*: We consider both *exact* consensus [33, 25] and *approximate* consensus [15, 16]. Exact consensus requires the fault-free nodes to agree on exactly the *same* output; whereas, approximate consensus requires the fault-free nodes to produce outputs within a certain constant ϵ ($\epsilon > 0$) of each other.

- *Fault Model*: In this article, we present only work on node faults.[1] In the fault models, all links are assumed to be reliable. However, nodes may suffer crash or Byzantine faults. Crash fault assumes fail-stop failure model; whereas, Byzantine nodes may behave arbitrarily, including sending incorrect and mismatching (or inconsistent) messages to different neighbors. Here, we consider the scenario where up to f nodes in the system may be faulty (f-total fault model).[2]

- *System Synchrony*: In synchronous systems, nodes proceed in a lock-step fashion, and we consider both exact and approximate consensus. In asynchronous systems, no known bound on the communication delay or processing speed exists. Moreover, it is known that exact consensus is impossible to achieve in asynchronous systems [18]; hence, we consider only approximate consensus.

- *Algorithm Type*: We consider two types of consensus algorithms:

  - *General algorithms*: (i) nodes have complete knowledge of the network topology, and can route messages to other nodes; and (ii) no constraint is placed on the amount of state a node may maintain.
  - *Iterative algorithms*: (i) nodes proceed in iterations; (ii) the computation of a new state at each node is based only on local information, i.e., the node's own state and states of the neighboring nodes; and (iii) after each iteration of the algorithm, the state of each fault-free node must remain in the convex hull of the states of the fault-free nodes at the end of the previous iteration.

[1] Please refer to [38, 41] for discussion on work focusing on link faults.

[2] Please refer to [38] for discussion about work on f-local fault model (up to f incoming nodes of a fault-free node may be faulty) [39, 49, 29], and generalized fault model (a fault model specifying potential locations of failures) [43, 19].

General algorithms usually achieve consensus more efficiently, e.g., [3, 30, 33, 25]; however, they requires extra information and overhead, such as topology knowledge and routing mechanism. Therefore, iterative algorithms have been studied extensively as well. Byzantine fault-tolerant iterative consensus has been well explored in complete and undirected graphs, e.g., [15, 1, 3, 30, 17]. Using iterative algorithms to achieve fault-free consensus [5, 22] and approximate consensus in a restricted fault model [27, 28, 49, 29] have been studied extensively as well.

Problem Formulation Here, we introduce different problem formulations that will be discussed in this article. That is, a correct consensus algorithm must satisfy the corresponding properties below. There are some differences between problem formulations for crash-tolerant and Byzantine algorithms, because for crash faults, we do not mind if the output is an input of some crashed nodes, whereas, for Byzantine faults, we would like to avoid using Byzantine input as the output.

- *Exact Crash-Tolerant Consensus*: (i) **Agreement**: the output (i.e., decision) at all the fault-free nodes is identical; (ii) **Validity**: the output of each fault-free node equals the input of one of the nodes; and (iii) **Termination**: every fault-free node decides on an output in finite amount of time. For exact consensus, we assume that the input is an integer in $[0, K]$ for some positive K.

- *Exact Byzantine Consensus with Binary Input*: (i) **Agreement**: the output (i.e., decision) at all the fault-free nodes is identical; (ii) **Validity**: the output of every fault-free node equals the input of a fault-free node; (iii) **Termination:** every fault-free node decides on an output in finite amount of time.

- *Exact Byzantine Consensus with Multi-Valued Input*: (i) **Weak Validity** [24]: If all fault-free nodes have the same input, then the output of every fault-free node equals its input; (ii) **Agreement**, and (iii) **Termination** (same as binary version).

- *Iterative Approximate Crash-Tolerant Consensus*: (i) **Validity**: After each iteration of an iterative algorithm, the state of each fault-free node must remain in the *convex hull* of the inputs of all the nodes; and (ii) **Convergence**: For any $\epsilon > 0$, after a sufficiently large number of iterations, the states of the fault-free nodes are guaranteed to be within ϵ of each other.

- *Iterative Approximate Byzantine Consensus*: (i) **Validity**: After each iteration of an iterative algorithm, the state of each fault-free node must remain in the *convex hull* of the inputs of the fault-free nodes; and (ii) **Convergence** (same as crash-tolerant version).

1.3 Summary of Results in Directed Networks

Given the four dimensions and consensus problems discussed in Section 1.2, we are ready to present the summary of work on directed networks. As addressed in Table 1, tight conditions on the undirected networks for solving different types of consensus problems have been identified. In addition, the tight condition for solving approximate crash-tolerant consensus in synchronous systems follows from the decentralized control literature [5, 22]. Some recent papers address the following problems.

- *General algorithms*:

Table 2: Summary of Recent Results on Directed Networks

Fault Model	**System**	**Output**	**General Alg.**	**Iterative Alg.**
Crash	Synchronous	Exact	[42]	
		Approx.	Follows from [5, 22]	
	Asynchronous	Approx.	[42]	[38]
Byzantine	Synchronous	Exact	[42]	Open
		Approx.	[37]	[46]
	Asynchronous	Approx.	Open	[46]

1. Exact crash-tolerant consensus in synchronous systems [42]
2. Approximate crash-tolerant consensus in asynchronous systems [42]
3. Exact Byzantine consensus in synchronous systems [42]
4. Approximate Byzantine consensus in synchronous systems [37]

- *Iterative algorithms*:

1. Approximate Byzantine consensus in both synchronous and asynchronous systems [46]
2. Approximate crash-tolerant consensus in asynchronous systems [38]

There are two problems that remain open:

- Using general algorithms to solve approximate Byzantine consensus in *asynchronous* systems, and
- Using iterative algorithms to solve exact Byzantine consensus in synchronous systems.

Table 2 summarizes results on consensus in directed networks and open problems in this area. We discuss other related work in directed networks in Section 5.

2 Preliminary

2.1 System Model

The model is typical in the literature [30, 3]. We briefly discuss it for completeness. We consider a point-to-point message-passing network in which nodes are connected by *directed* links. The communication network is *static*, and it is represented by 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 directed edge (i, j) is in $\mathcal{E}$. Each node can send messages to itself as well; however, for convenience, we exclude self-loops from set $\mathcal{E}$. We will often use the terms *edge* and *link* interchangeably. All the communication links are assumed to be reliable, FIFO (first-in first-out) and deliver each transmitted message exactly once.

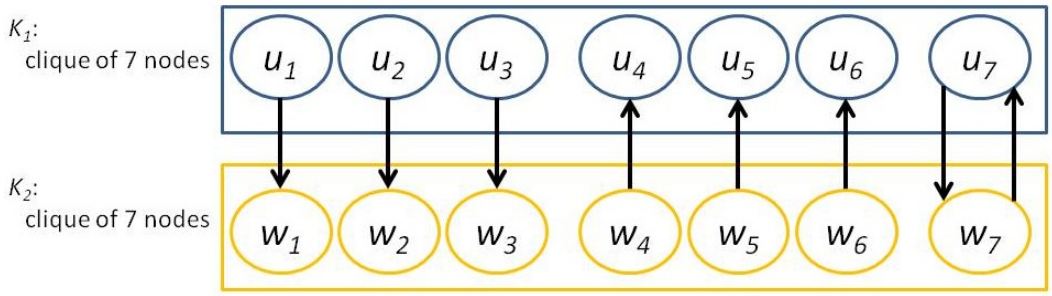


Figure 4: Edges within cliques K_1 and K_2 are not shown.

2.2 Terminology

Here, we introduce some terminology to facilitate the discussion. Upper case letters are used to name sets. Lower case italic letters are used to name nodes. All paths used in our discussion are directed paths. Recall that we consider the directed graph $G(\mathcal{V}, \mathcal{E})$. In G, node j is said to be an incoming neighbor of node i if $(j, i) \in \mathcal{E}$. Let N_i^- be the set of incoming neighbors of node i, i.e., $N_i^- = \{j \mid (j, i) \in \mathcal{E}\}$. Define N_i^+ as the set of outgoing neighbors of node i, i.e., $N_i^+ = \{j \mid (i, j) \in \mathcal{E}\}$. For set $B \subseteq \mathcal{V}$, node i is said to be an incoming neighbor of set B if $i \notin B$, and there exists $j \in B$ such that $(i, j) \in \mathcal{E}$. Given subsets of nodes A and B, set B is said to have k incoming neighbors in set A if A contains k distinct incoming neighbors of B. We now introduce two notations that are used frequently.

Definition 1. *Given disjoint non-empty subsets of nodes A and B, $A \overset{x}{\longmapsto} B$ if B has at least x distinct incoming neighbors in A. When it is not true that $A \overset{x}{\longmapsto} B$, we will denote that fact by $A \overset{x}{\not\longmapsto} B$.*

Consider the network in Figure 4, which contains two cliques K_1 and K_2, each consisting of 7 nodes. Within each clique, each node has a directed link to the other 6 nodes in that clique — these links within each clique are not shown in the figure. There are 8 directed links with one endpoint in clique K_1 and the other endpoint in clique K_2. In the network, K_2 has 4 incoming neighbors in K_1, namely u_1, u_2, u_3 and u_4. Thus, $K_1 \overset{4}{\longmapsto} K_2$. Similarly, $K_2 \overset{4}{\longmapsto} K_1$.

Definition 2. *Given disjoint non-empty subsets of nodes A and B, $A \overset{x}{\Rightarrow} B$ if each node in B has at least x distinct incoming neighbors in A. When it is not true that $A \overset{x}{\Rightarrow} B$, we will denote that fact by $A \overset{x}{\not\Rightarrow} B$.*

$\Rightarrow$ is different from $\longmapsto$ in Definition 1. $A \Rightarrow B$ means that *each* node in B has enough incoming neighbors that are in A; whereas, $A \longmapsto B$ means that all nodes in B *jointly* have enough incoming neighbors that are in A. Notations $\longmapsto$ and $\Rightarrow$ are used to specify conditions for general and iterative algorithms, respectively. In Figure 4, $K_1 \overset{1}{\not\Rightarrow} K_2$, since w_4, w_5, and w_6 do not have any incoming neighbor in set K_2.

2.3 Main Results

We summarize the results from [42, 46, 37, 38] below. Recall that the problem formulations were introduced in Section 1.2.

General Algorithms We first list the results for solving different consensus problems using general algorithms. We will use the notation $\longmapsto$ defined in Definition 1 to specify the tight conditions. Each theorem below requires the graph to satisfy a certain condition: we name the conditions presented in Theorems 1, 2 and 3 as **CCS-G** (abbreviating Crash-Consensus-Synchronous-General), **CCA-G** (Crash-Consensus-Asynchronous-General) and **BCS-G** (Byzantine-Consensus-Synchronous-General), respectively. Characterization of the necessary and sufficient condition on the network topology for using general algorithms to solve approximate Byzantine consensus in asynchronous systems remains open.

Theorem 1. *Exact and approximate crash-tolerant consensus in a synchronous system are solvable using a general algorithm iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{1}{\longmapsto} R$ or $R \cup C \overset{1}{\longmapsto} L$.*
(Condition CCS-G)

Theorem 2. *Approximate crash-tolerant consensus in an asynchronous system is solvable using a general algorithm iff for any partition L, C, R of $\mathcal{V}$, where L and R are both non-empty, either $L \cup C \overset{f+1}{\longmapsto} R$ or $R \cup C \overset{f+1}{\longmapsto} L$.*
(Condition CCA-G)

Theorem 3. *Exact and approximate Byzantine consensus in a synchronous system are solvable using a general algorithm iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{f+1}{\longmapsto} R$ or $R \cup C \overset{f+1}{\longmapsto} L$.*
(Condition BCS-G)

The network shown in Figure 4 above satisfies Condition BCS-G for $f = 2$, whereas the network in Figure 3 above satisfies Condition CCS-G for $f = 1$. Note that the condition in Claim 1 is indeed equivalent to Condition CCS-G; thus, the claim is implied by Theorem 1. We will discuss the equivalence later.

Iterative Algorithms Below, we list the results for solving different consensus problems using iterative algorithms. We will use the notation $\Rightarrow$ defined in Definition 2 to specify the tight conditions. Each theorem below requires the graph to satisfy a certain condition: we name the conditions presented in Theorems 4, 5, 6 and 7 as **CCS-I** (abbreviating Crash-Consensus-Synchronous-Iterative), **CCA-I** (Crash-Consensus-Asynchronous-Iterative), **BCS-I** (Byzantine-Consensus-Synchronous-Iterative), and **BCA-I** (Byzantine-Consensus-Asynchronous-Iterative), respectively. Characterization of the necessary and sufficient condition for using iterative algorithms to solve exact Byzantine consensus in synchronous systems remains open. Note that Condition CCS-G is identical to Condition CCS-I; however, all the other conditions for synchronous and asynchronous systems are not equal to their counterparts.

Theorem 4. *Exact and approximate crash-tolerant consensus in a synchronous system are solvable using an iterative algorithm iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{1}{\Rightarrow} R$ or $R \cup C \overset{1}{\Rightarrow} L$.*
(Condition CCS-I)

Theorem 5. *Approximate crash-tolerant consensus in an asynchronous system is solvable using an iterative algorithm iff for any partition L, C, R of $\mathcal{V}$, where L and R are both non-empty, either*

$L \cup C \overset{f+1}{\Rightarrow} R$ *or* $R \cup C \overset{f+1}{\Rightarrow} L$.
(Condition CCA-I)

Theorem 6. *Approximate Byzantine consensus in a synchronous system is solvable using an iterative algorithm iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{f+1}{\Rightarrow} R$ or $R \cup C \overset{f+1}{\Rightarrow} L$.*
(Condition BCS-I)

Theorem 7. *Approximate Byzantine consensus in an asynchronous system is solvable using an iterative algorithm iff for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \leq f$, either $L \cup C \overset{2f+1}{\Rightarrow} R$ or $R \cup C \overset{2f+1}{\Rightarrow} L$.*
(Condition BCA-I)

Note that in Condition CCA-G (in Theorem 2) and Condition CCA-I (in Theorem 5), the partition does not need set F, unlike other conditions.

Intuition For consensus to be achieved, there must be a way for information to "flow between" different subsets of fault-free nodes (subsets L and R in the theorems above), despite the presence of faulty nodes (subset F). The different conditions above capture this intuition. Observe that, in each case, for different values of x, we obtain the requirement of the form "either $L \cup C \overset{x}{\longmapsto} R$ or $R \cup C \overset{x}{\longmapsto} L$" (or analogously $L \cup C \overset{x}{\Rightarrow} R$ or $R \cup C \overset{x}{\Rightarrow} L$). Intuitively, after removing the subset F (i.e., isolating the faulty behavior), information must be able to "flow" either from $L \cup C$ to R, or from $R \cup C$ to L in the remaining graph, but it is not necessary that the information flows in both directions — this "asymmetry" in the necessary and sufficient condition is a consequence of the directed nature of the communication network. The value of x is to ensure that some fault-free node(s) in either L or R has enough redundant information/messages to mask faulty behaviors.

For general algorithms, we use $\longmapsto$, because with the topology information and the ability to route message, nodes in the set R or L may be able to share information within the set. In contrast, for iterative algorithms, computation is performed locally, and thus, we use $\Rightarrow$ to represent the tight condition. That is, every node requires enough incoming neighbors from outside. Last, in many cases, we are able to obtain an equivalent condition that requires the existence of directed rooted spanning tree in the reduced graph. For example, the condition in Claim 1 is equivalent to Condition CCS-G (in Theorem 1).[3]

Lower Bounds on Number of Nodes As shown below, the tight conditions imply the lower bounds on the number of nodes, n. Recall that by definition, $|\mathcal{V}| = n$. These lower bounds are well-known results in complete and undirected graphs [3, 30, 17, 15, 33]. We include them here for completeness. Recall that by assumption, up to f nodes may be faulty in the system.

- Condition CCA-G implies that $n \geq 2f + 1$.
- Condition BCS-G implies that $n \geq 3f + 1$.
- Condition CCA-I implies that $n \geq 2f + 1$.

[3] As shown in [46, 42, 37], the manner of constructing reduced graphs varies for different consensus problems. Claim 1 only provides the construction for crash-tolerant consensus in synchronous systems.

- Condition BCS-I implies that $n \geq 3f + 1$.
- Condition BCA-I implies that $n \geq 5f + 1$.

The proofs for the claims above can be found in [38, 46].

Relations among Tight Conditions As noted above, these tight conditions capture how information can "flow between" different subsets of fault-free nodes despite the presence of faulty nodes under different synchrony assumptions. This section compares these tight conditions.

Lemma 1. *Conditions CCS-G, CCA-G and BCS-G are progressively stronger.*

- *Condition BCS-G implies Condition CCA-G, but not vice-versa.*
- *Condition CCA-G implies Condition CCS-G, but not vice-versa.*

The proof can be found in [42]. Similarly, the following lemmas can be proved.

Lemma 2. *Conditions CCS-I, CCA-I, BCS-I and BCA-I are progressively stronger.*

- *Condition BCA-I implies Condition BCS-I, but not vice-versa.*
- *Condition BCS-I implies Condition CCA-I, but not vice-versa.*
- *Condition CCA-I implies Condition CCS-I, but not vice-versa.*

It turned out that only are Conditions CCS-I and CCS-G equivalent, and the rest pairs of conditions for iterative and general algorithms are all different as stated in the lemma below.

Lemma 3.

- *Condition CCA-I implies Condition CCA-G, but not vice-versa.*
- *Condition BCS-I implies Condition BCS-G, but not vice-versa.*

3 Crash-Tolerant Consensus in Synchronous Systems

In this section, we discuss results related to solving exact crash-tolerant consensus in synchronous systems using *general algorithms* in detail. Recall that we defined the crash-tolerant consensus problems in Section 1.2. Then, we briefly discuss other results related to crash-tolerant consensus in synchronous systems.

3.1 Solving Exact Consensus using General Algorithms

By definition, general algorithms allow nodes to have complete knowledge of network topology; hence, it is possible to use a routing mechanism in general algorithms.

3.1.1 Necessity of Condition CCS-G

Theorem 1 in Section 2.3 presented the necessary and sufficient condition (named Condition CCS-G) for solving the above problem in directed graphs using general algorithms. It is straightforward why Condition CCS-G is necessary, since to achieve exact consensus, some shared information needs to "flow" to any two subsets of fault-free nodes after some other nodes crash. If Condition CCS-G does not hold for the graph G, then it is possible to find a subset of up to f nodes whose removal would "block" the flow of shared information to two sets of other nodes.

3.1.2 Sufficiency of Condition CCS-G (Exact Consensus)

In [42], we proved the sufficiency of Condition CCS-G constructively by presenting an algorithm, called MVC, and proving its correctness. Algorithm MVC can achieve consensus with multi-valued inputs, i.e., an input is an integer in $[0, K]$ for some positive K. MVC uses Algorithm Min-Max presented below as a component. Algorithm Min-Max can achieve consensus with *binary* inputs (0 or 1). These two algorithms prove sufficiency of Condition CCS-G, but they are not necessarily the most efficient. Development of optimal algorithms needs further research. For brevity, we only discuss the binary consensus algorithm below. Interested readers can find Algorithm MVC in [42].

Algorithm Min-Max

Note that Algorithm *Min-Max* has input parameter x_i. To achieve binary consensus, each node i performs Algorithm *Min-Max* passing its binary input value as parameter x_i to Algorithm *Min-Max*. Algorithm Min-Max uses *Compute* as a sub-routine. *Compute* has two parameters: t, which is a binary value, and *Function*, which may be specified as *Min* and *Max*. In the last step of each round in *Compute* at node i, the *Function* is applied to set S_i. $Min(S_i)$ returns the minimum of the values in set S_i, and $Max(S_i)$ returns the maximum of the values in set S_i.

Algorithm Min-Max(x_i) for node $i \in \mathcal{V}$

Initialization: $v_i[0]$:= parameter x_i passed to Min-Max

- For phase number $p := 1$ to $2f + 2$:

  If p *mod* $2 = 0$, then (**Min Phase**)

  $v_i[p]$:= Compute ($v_i[p-1]$, *Min*)

  Else, (**Max Phase**)

  $v_i[p]$:= Compute ($v_i[p-1]$, *Max*)

- Return $v_i[2f + 2]$

Compute(t, *Function*) for node $i \in \mathcal{V}$

- $\tau_i := t$
- Perform $n - 1$ rounds, each round consisting of the four steps below:

  Send τ_i to all the nodes in $N_i^+ \cup \{i\}$

  Receive values from $N_i^- \cup \{i\}$

Denote the set of values received
in the previous step as S_i
$\tau_i := Function(S_i)$

- Return τ_i

Correctness of Algorithm Min-Max with Binary Inputs We first prove a useful lemma, and introduce the notion of *source* of the graph.

Lemma 4. *Suppose that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS-G. For any $F \subseteq \mathcal{V}$, such that $|F| \leq f$, let G_F denote the subgraph of G induced by the nodes in $\mathcal{V} - F$. There exists at least one node in G_F that has directed paths in G_F to all nodes in $\mathcal{V} - F$. Such a node is said to be a source for G_F.*

Proof. The proof of the lemma is by contradiction. Suppose that Graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS-G, and for some $F \subseteq \mathcal{V}$, $|F| \leq f$, there exists a pair of nodes $i, j \notin F$ such that there is no node s that has directed paths to both i and j in subgraph G_F induced by nodes in $\mathcal{V} - F$. For the subgraph G_F and a node x in $\mathcal{V} - F$, define S_x as the set of all nodes that have directed paths in G_F to node x. Note that S_x contains x as well, because x trivially has a path to itself.

By assumption, S_i and S_j are disjoint. Moreover, there must be no path from any node in S_i to any node in S_j in G_F, and vice versa, since otherwise, there would exist some node that can reach both nodes i and j, which contradicts our assumption above. Now, define L, C, R as follows:

- $L := S_i$
- $R := S_j$
- $C := \mathcal{V} - F - L - R$

Then, we make the following observations:

- *F and C may be empty, but L and R are non-empty*: This is true because $i \in S_i = L$ and $j \in S_j = R$.
- *Nodes in C (if non-empty) have no link to nodes in $L \cup R$*: If some node $c \in C$ has a link to some node $x \in L = S_i$, then c will be able to reach node i on a directed path via node x (since $x \in S_i$ has a path to i, by definition of set S_i). This would then imply that c must be in S_i, however, that contradicts the definition of C as $\mathcal{V} - F - L - R$. By a similar argument, nodes in C cannot have links to nodes in R.
- *There is no link from any node in L to nodes in R, and vice versa*: Recall that $L = S_i$ and $R = S_j$. If some node $x \in L$ has a link to a node $y \in R$, then x will have a directed path to node j via node y. However, this contradicts our assumption above that no node has directed paths to both i and j.

These observations together imply that $L \cup C \overset{1}{\not\longmapsto} R$ and $C \cup R \overset{1}{\not\longmapsto} L$. That is, $G(\mathcal{V}, \mathcal{E})$ does not satisfy Condition CCS-G. This is a contradiction. Thus, Lemma 4 is proved. □

From the proof above, it is not hard to prove the other direction, which implies that Condition CCS-G is equivalent the condition in Claim 1. The proof is left as an exercise for readers. Lemma 4 defines the notion of a *source* node. Essentially, the lemma shows the existence of a *directed rooted spanning tree* in the induced graph G_F, which has the source node as the root. Note that the induced graph G_F is also called "reduced graph" [42]. Presence of such "source" nodes (or root) is crucial in achieving consensus. Similar observations were first made in the context of fault-free consensus [5, 22].

Now, we are ready to show the correctness of Algorithm Min-Max. The proof of correctness assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS-G.

Lemma 5. *Algorithm Min-Max satisfies the termination, agreement and validity properties.*

Proof. Since Algorithm Min-Max executes a fixed number of phases, its termination occurs in finite time. Validity is satisfied trivially as well. Now we prove that the algorithm satisfies the agreement property when the inputs are binary (0 or 1). We start by observing that $Compute(t, Min)$ never returns a value larger than parameter t passed to *Compute*, and $Compute(t, Max)$ never returns a value smaller than parameter t passed to *Compute*.

Fix an execution of the algorithm. Since there are $2f + 2$ phases, there must exist a pair of consecutive phases p^*, $p^* + 1$ such that *no node crashes* in phases p^* and $p^* + 1$. Without loss of generality, let p^* be the Min Phase (i.e., p^* *mod* $2 = 0$) and $p^* + 1$ be the Max Phase. Denote by F the set of nodes that crash before starting phase p^*.

Lemma 4 shows the existence of a source node that has directed paths in G_F to all nodes in $\mathcal{V} - F$ (G_F is defined in the Lemma). In general, there may be multiple such source nodes in G_F. Consider the two cases below. In each case, we show that agreement is achieved.

- *Case I: There exists a source s in G_F for which $v_s[p^* - 1] = 0$*: Thus, during the Min Phase p^*, node s will call *Compute(0,Min)*. Then during the first round of *Compute* in phase p^*, those nodes in $\mathcal{V} - F$ with incoming links from node s will update their τ variable (within *Compute*) to be 0. Recall that we are presently assuming binary inputs. Since the source node has directed paths (of length at most $n - 1$) to all the nodes in G_F, it follows by induction that each node i in $\mathcal{V} - F$ will update its state τ_i to be 0 by the end of the $n - 1$ rounds performed within *Compute*. Thus, when *Compute* returns, $v_i[p^*]$ at each $i \in \mathcal{V} - F$ will be set to 0. It should be easy to see that the remaining phases will not change the value of v_i at the fault-free nodes, ensuring agreement when the algorithm terminates.

- *Case II: For each source s in G_F, $v_s[p^* - 1] = 1$*:
  In this case, we argue that, for each source s, $v_s[p^*] = 1$. Suppose, by way of contradiction, that each source s of G_F has $v_s[p^* - 1] = 1$, but there exists a source node s' for which $v_{s'}[p^*] = 0$. For this to happen, node s' must receive 0 on a path from some other *non-source* node z during phase p^*. This implies that $v_z[p^* - 1] = 0$; additionally, the fact that there exists a path in G_F from z to the source node s' implies that z is also a source in G_F. This contradicts the assumption that all source nodes in G_F have state equal to 1 at the start of phase p^*.

  This shows that, for each source node s, we have $v_s[p^*] = 1$. Now consider Max Phase $p^* + 1$. Recall that no node crashes in Phases p^* and $p^* + 1$. Thus, by an argument analogous to that used for Min Phase p^* in Case I above, it follows that, for all $i \in \mathcal{V} - F$, $v_i[p^* + 1] = 1$,

achieving agreement. Any additional phases beyond phase $p^* + 1$ will not result in violation of the agreement, similar to Case I.

□

As addressed above, we have developed a multi-valued algorithm, Algorithm MVC, in [42]. This proves the sufficiency of Condition CCS-G for achieving exact consensus. We defer the discussion of approximate consensus to Section 3.2.2.

3.2 Other Results

In this section, we discuss other results for solving crash-tolerant consensus in synchronous systems.

3.2.1 Solving Consensus using Iterative Algorithms

We presented a general consensus algorithm in Section 3.1. Here, we consider iterative approximate consensus (defined in Section 1.2).

Necessity and Sufficiency of Condition CCS-I (Approximate Consensus) Theorem 4 in Section 2.3 presented the necessary and sufficient condition (named Condition CCS-I) for solving approximate consensus using iterative algorithms. Necessity is trivial, since Condition CCS-I is equivalent to CCS-G. Moreover, since Condition CCS-I implies the existence of directed rooted spanning tree at all time (Claim 1 and Lemma 4), it is straightforward to adapt the analysis for iterative fault-free consensus algorithms to show that Condition CCS-I is sufficient for achieving approximate crash-tolerant consensus [5, 22]. Lately, Charron-Bost et al. extended the analysis to dynamic directed graphs [11].

Necessity and Sufficiency of Condition CCS-I (Exact Consensus) It is straightforward to adapt Algorithm Min-Max to iterative structure, since we do not require any routing mechanism in Algorithm Min-Max. Moreover, the state at each fault-free node is guaranteed to be some fault-free node's input (as shown in proof of Lemma 5). Thus, the iterative version of Algorithm Min-Max solves binary exact consensus. However, Algorithm MVC, the multi-valued consensus algorithm based on Algorithm Min-Max, relies on more complicated state variables, and does not satisfy the validity property mentioned in Section 3.2.1. The problem of using iterative algorithm to achieve multi-valued exact consensus remains open.

3.2.2 Solving Approximate Consensus using General Algorithms

We discussed an exact consensus algorithm, Algorithm MVC, in Section 3.1. The existence of Algorithm MVC only partially proves Theorem 4, since Algorithm MVC does not solve approximate consensus — approximate consensus allows input to be an arbitrary real number in $[0, K]$; whereas, in exact consensus, the input is restricted to some integer in $[0, K]$. To see that Condition CCS-G is also sufficient for solving approximate consensus, observe that Condition CCS-I is equivalent to CCS-G; hence, Theorem 4 implies the claim of tight condition on the directed communication networks for achieving approximate consensus in Theorem 1.

4 Byzantine Consensus in Synchronous Systems

Section 3 presented results related to crash faults. Here, we study Byzantine faults. For brevity, we only present the intuition in this section, and the full proofs and the algorithms are presented in [40, 38, 37]. Recall that we defined the Byzantine consensus problems in Section 1.2.

4.1 Solving Exact Consensus using General Algorithms

4.1.1 Necessity and Sufficiency of Condition BCS-G

Theorem 3 in Section 2.3 presented the necessary and sufficient condition (named Condition BCS-G) for solving exact consensus problem in directed graphs using general algorithms. The necessity of Condition BCS-G can be proved using the indistinguishability proofs [17, 13]. To prove sufficiency, we first develop an algorithm to achieve Byzantine consensus with *binary* inputs, and then use it to achieve multi-valued Byzantine consensus. We now introduce the notion of *reduced graph* and then present an observation that is useful to construct a Byzantine consensus algorithm in directed graphs that satisfy Condition BCS-G. Note that the notion of reduced graph here is different from the one presented in Claim 1 and Lemma 4.

Definition 3. (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$, and sets $F \subset \mathcal{V}$, $F_1 \subset \mathcal{V} - F$, such that $|F| \leq f$ and $|F_1| \leq f$, reduced graph $G_{F,F_1}(\mathcal{V}_{F,F_1}, \mathcal{E}_{F,F_1})$ is defined as follows: (i) $\mathcal{V}_{F,F_1} = \mathcal{V} - F$, and (ii) $\mathcal{E}_{F,F_1}$ is obtained by removing from $\mathcal{E}$ all the links incident on the nodes in F, and all the outgoing links from nodes in F_1. That is, $\mathcal{E}_{F,F_1} = \mathcal{E} - \{(i,j) \mid i \in F \text{ or } j \in F\} - \{(i,j) \mid i \in F_1\}$.*

Intuition Suppose $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS-G. Then, for any $F \subset \mathcal{V}$ and $F_1 \subset \mathcal{V} - F$, where $|F| \leq f$ and $|F_1| \leq f$, there exists a set of at least $f+1$ nodes $S \subseteq \mathcal{V} - F$ such that (i) nodes in S are strongly connected in reduced graph G_{F,F_1}, and (ii) for each $j \in \mathcal{V} - F - S$, there exist at least $f+1$ pairwise node-disjoint paths from S to j in G that do not contain any nodes in F.

The proposed algorithm for binary consensus maintains a state variable at each node, with the invariant that this state variables at each fault-free node aways has a "valid" value, where a value is "valid" if it is an input at some fault-free node. Intuitively, the above $f+1$ disjoint paths from the nodes in S to nodes in $\mathcal{V} - F - S$ provide adequate redundancy to allow propagation of values from the nodes in S to the nodes in $\mathcal{V} - F - S$, with the guarantee that any potential message tampering by faulty nodes would not cause the recipients to accept an "invalid" value. In the fortuitous event that the nodes in F are faulty and the nodes in S (which are fault-free) have the same valid state variable, this value is then propagated to all the fault-free nodes, achieving consensus. The algorithm ensures that this fortuitous event occurs at least once during the execution. The rest of the details are presented in [40, 38].

Interesting Observation One interesting implication of Condition BCS-G is that there exist graphs that satisfy Condition BCS-G wherein reliable communication may *not* be feasible in either direction across a given cut. For a pair of fault-free nodes x, y, reliable communication is feasible from x to y if there are $2f+1$ node-disjoint directed paths from x to y (when up to f nodes may fail). Now, consider the network in Figure 4 again. Observe that there are only 4 directed links from K_1 to K_2, and 4 directed links from K_2 to K_1. Thus, reliable communication is *not guaranteed* across the cut (K_1, K_2) in either direction when $f = 2$ (Byzantine faults). Yet, Byzantine consensus

is achievable in synchronous systems since this graph satisfies Condition BCS-G for $f = 2$. We prove this claim in [40, 38].

4.2 Solving Approximate Consensus using General Algorithms

4.2.1 Necessity and Sufficiency of Condition BCS-G

In [37], Su and Vaidya introduced a family of algorithms in which nodes can perform routing within their l-hop neighborhood $(1 \leq l \leq d)$, where d is the diameter of the graph, and proved the tight condition for each choice of l. Note that when $l = 1$, the algorithm belongs to the class of iterative algorithms, and when $l = d$, the algorithm belongs to the class of general algorithms. The second part of Theorem 3 follows from the results in [37].

4.3 Solving Approximate Consensus using Iterative Algorithms

Here, we briefly discuss the results on approximate Byzantine consensus, i.e., Theorem 6. Note that as addressed before, the tight condition for using iterative algorithms to solve exact Byzantine consensus in synchronous systems remains open.

4.3.1 Necessity and Sufficiency of Condition BCS-I

In [46], we proved that Condition BCS-I is necessary using logic similar to the indistinguishability proofs [17, 13]. Roughly speaking, if Condition BCS-I does not hold, then we can let the nodes in F (if non-empty) to be Byzantine faulty, and there exists a certain Byzantine behavior to make sure no algorithm satisfies ϵ-agreement and validity at the same time. For sufficiency, we provided an iterative algorithm in [46], which was inspired by the iterative algorithm developed for complete networks [15]. To discuss the intuition of the correctness, let us present another type of reduced graph. Note that the notion of reduced graph here is different from Definition 3.

Definition 4. (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$ and $F \subset \mathcal{V}$, a graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ is said to be a* reduced graph*, if: (i) $\mathcal{V}_F = \mathcal{V} - F$, and (ii) $\mathcal{E}_F$ is obtained by first removing from $\mathcal{E}$ all the links incident on the nodes in F, and* then *removing up to f other incoming links at each node in $\mathcal{V}_F$.*

Intuition Condition BCS-I can be proved to be equivalent to the following condition: *in all reduced graphs as per Definition 4, there exists a directed rooted spanning tree.* The complete proof is presented in [46]. We developed a proof technique based on famous matrix tools [22, 48, 5, 20] to prove the correctness of fault-tolerant iterative algorithms [44, 43, 45]. We used the technique along with the equivalent condition to prove that the proposed iterative algorithm correctly achieves approximate Byzantine consensus in synchronous systems in [44]. This shows that Condition BCS-I is sufficient.[4]

5 Related Work

Consensus with Various Graphs Properties Previous work also studied graph properties for other related problems. Bansal et al. [4] identified tight conditions for achieving exact Byzantine

[4]The sufficiency of Condition BCS-I can also be proved using first-principle approach as presented in [46].

consensus with *authentication* tools in *undirected* graphs. Bansal et al. discovered that all-pair reliable communication is not necessary to achieve consensus when using authentication. This article discusses results that do not rely on authentication tools. As discussed in Section 4.1.1, all-pair reliable communication is not necessary for consensus in this case. Alchieri et al. [2] explored the problem of achieving exact consensus in *unknown* networks with Byzantine nodes, but the underlying communication graph is assumed to be *fully-connected* (complete network). The results presented in this article assume either the knowledge of immediate neighbors (iterative algorithms), or the knowledge of the complete topology. Moreover, the communication network may be incomplete.

Recently, researchers explored the consensus problem in directed dynamic networks [9, 8, 10, 11, 35], where communication network changes over time. For synchronous systems, [10, 11] solved *approximate* crash-tolerant consensus in directed dynamic networks using local averaging algorithms. In the asynchronous setting, [10, 11] addressed approximate consensus with crash faults in *complete* graphs (which are necessarily undirected). [9, 8, 10, 11, 35] did not consider Byzantine faults.

[9, 35, 8] considered the *message adversary*, which controls the communication pattern, i.e., the adversary has the power to specify the sets of communication graphs. Biely et al. studied the exact consensus problem [8] and k-set consensus problem [9, 35] (i.e., at most k different outputs at fault-free nodes) in dynamic networks under the *message adversary*, and the system is assumed to be synchronous. All the nodes are assumed to be fault-free in [9, 35, 8].

Iterative Approximate Consensus in Incomplete Graphs There is also rich work on using iterative algorithms to solve approximate consensus in the presence of faults. Dolev et al. presented the early results on Byzantine fault-tolerant iterative consensus [15]. The initial algorithms [15, 30] were proved correct in *fully connected* networks (i.e., complete networks). Fekete [16] studied the convergence rate of the approximate consensus algorithms. Abraham et al. proposed an algorithm for approximate Byzantine consensus [1] that has optimal resilience (optimal number of nodes for achieving consensus).

A restricted fault model — called "malicious" fault model — in which the faulty nodes are restricted to sending identical messages to their neighbors has also been explored extensively [27, 28, 49, 29]. In contrast, the Byzantine model considered in this article allows a faulty node to send different messages to different neighbors. LeBlanc and Koutsoukos [27] addressed a continuous time version of the consensus problem with malicious faults in complete graphs. Under both malicious and Byzantine fault models, LeBlanc and Koutsoukos [28] have identified some sufficient conditions under which the continuous time version of iterative consensus can be achieved with up to f faults in the network; however, these sufficient conditions are *not* tight.

For the malicious fault model, LeBlanc et al. [29] have obtained *tight* necessary and sufficient conditions for tolerating up to f total number of faults in the network (f-total fault model). Under the malicious model, since a faulty node must send identical messages to all the neighbors, the necessary and sufficient conditions are weaker than those developed for the Byzantine fault model (in [46]). For instance, under the malicious model, iterative consensus is possible in a complete graph consisting of $2f + 1$ nodes, whereas at least $3f + 1$ nodes are necessary for consensus under the Byzantine fault model.

Iterative approximate consensus algorithms that do not tolerate faulty behavior have been studied extensively in the decentralized control area. Bertsekas and Tsitsiklis [5] and Jadbabaei, Lin and Morse [22] have explored approximate consensus in the absence of faults, using only near-

neighbor communication in systems wherein the communication graph may be partially connected and dynamic.

Reliable Communication and Broadcast Several papers have also addressed communication between a single source-receiver pair, i.e., reliable communication problem. Dolev et al. [14] studied the problem of secure communication, which achieves both fault-tolerance and perfect secrecy between a single source-receiver pair in undirected graphs, in the presence of node and link failures. Desmedt and Wang considered the same problem in directed graphs [12]. Shankar et al. [36] investigated reliable communication between a source-receiver pair in directed graphs allowing for an arbitrarily small error probability in the presence of a Byzantine failures. Maurer et al. explored the problem in directed dynamic graphs [31].

There has also been works on the reliable broadcast problem, in which a fault-free source needs to send its input to all the fault-free nodes. The fault model under consideration is the f-local Byzantine fault model, in which up to f incoming neighbors of each fault-free nodes may become Byzantine faulty. [23] studied the problem in an infinite grid. [6, 7] developed a sufficient condition in the context of arbitrary network topologies, but the sufficient condition proposed is not tight. [34] provided necessary and sufficient conditions, but the two conditions are not tight either. [21] provided another condition that can approximate (within a factor of 2) the largest f for solving reliable broadcast. Independently, [32] and [39] presented the tight condition for achieving reliable broadcast in *undirected* and *directed* graphs, respectively.

6 Summary

In this article, we survey the results on fault-tolerant consensus in directed networks. We briefly discuss the intuitions on the results and some details of the tight conditions for synchronous systems. Finally, we discuss papers in message-passing networks that are relevant to this topic. Two problems in this area remain open:

- Using general algorithms to solve approximate Byzantine consensus in *asynchronous* systems, and
- Using iterative algorithms to solve exact Byzantine consensus in synchronous systems.

Acknowledgment

Research reported here is supported in part by the National Science Foundation.

References

[1] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239, 2004.

[2] E. Alchieri, A. Bessani, J. Silva Fraga, and F. Greve. Byzantine consensus with unknown participants. In T. Baker, A. Bui, and S. Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 22–40. Springer Berlin Heidelberg, 2008.

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

[4] P. Bansal, P. Gopal, A. Gupta, K. Srinathan, and P. K. Vasishta. Byzantine agreement using partial authentication. In *Proceedings of the 25th international conference on Distributed computing*, DISC'11, pages 389–403, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Optimization and Neural Computation Series. Athena Scientific, 1997.

[6] V. Bhandari and N. H. Vaidya. On reliable broadcast in a radio network. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, pages 138–147, New York, NY, USA, 2005. ACM.

[7] V. Bhandari and N. H. Vaidya. On reliable broadcast in a radio network: A simplified characterization. Technical report, University of Illinois at Urbana-Champaign, 2005.

[8] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2012.

[9] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *CoRR*, abs/1408.0620, 2014.

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

[11] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 528–539, 2015.

[12] Y. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 502–517. Springer Berlin Heidelberg, 2002.

[13] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1), March 1982.

[14] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the Association for Computing Machinery (JACM)*, 40(1):17–14, 1993.

[15] 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.

[16] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 73–87, New York, NY, USA, 1986. ACM.

[17] 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.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[19] V. K. Garg and J. Bridgman. The weighted byzantine agreement problem. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 524–531, May 2011.

[20] J. Hajnal. Weak ergodicity in non-homogeneous Markov chains. In *Proceedings of the Cambridge Philosophical Society*, volume 54, pages 233–246, 1958.

[21] A. Ichimura and M. Shigeno. A new parameter for a broadcast algorithm with locally bounded Byzantine faults. *Inf. Process. Lett.*, June 2010.

[22] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *Automatic Control, IEEE Transactions on*, 48(6):988 – 1001, june 2003.

[23] C.-Y. Koo. Broadcast in radio networks tolerating Byzantine adversarial behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 275–282, New York, NY, USA, 2004. ACM.

[24] L. Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, July 1983.

[25] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[26] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[27] H. LeBlanc and X. Koutsoukos. Consensus in networked multi-agent systems with adversaries. *14th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2011.

[28] H. LeBlanc and X. Koutsoukos. Low complexity resilient consensus in networked multi-agent systems with adversaries. *15th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2012.

[29] H. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications: Special Issue on In-Network Computation*, 31:766–781, April 2013.

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

[31] A. Maurer, S. Tixeuil, and X. Défago. Reliable communication in a dynamic network in the presence of Byzantine faults. *CoRR*, abs/1402.0121, 2014.

[32] A. Pagourtzis, G. Panagiotakos, and D. Sakavalas. Reliable broadcast with respect to topology knowledge. In *Proceedings of the 28th international conference on Distributed computing (DISC)*, 2014.

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

[34] A. Pelc and D. Peleg. Broadcasting with locally bounded Byzantine faults. *Inf. Process. Lett.*, 2005.

[35] M. Schwarz, K. Winkler, U. Schmid, M. Biely, and P. Robinson. Brief announcement: Gracefully degrading consensus and k-set agreement under dynamic link failures. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 341–343, New York, NY, USA, 2014. ACM.

[36] B. Shankar, P. Gopal, K. Srinathan, and C. P. Rangan. Unconditionally reliable message transmission in directed networks. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 1048–1055, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[37] L. Su and N. Vaidya. Reaching approximate Byzantine consensus with multi-hop communication. In A. Pelc and A. A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 9212 of *Lecture Notes in Computer Science*, pages 21–35. Springer International Publishing, 2015.

[38] L. Tseng. *Fault-Tolerant Consensus in Directed Graphs and Convex Hull Consensus*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[39] L. Tseng, N. Vaidya, and V. Bhandari. Broadcast using certified propagation algorithm in presence of Byzantine faults. *Information Processing Letters*, 115(4):512 – 514, 2015.

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

[41] L. Tseng and N. H. Vaidya. Iterative approximate consensus in the presence of Byzantine link failures. In *Networked Systems - Second International Conference, NETYS 2014, Marrakech, Morocco, May 15-17, 2014. Revised Selected Papers*, pages 84–98, 2014.

[42] L. Tseng and N. H. Vaidya. Fault-tolerant consensus in directed graphs. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 451–460, New York, NY, USA, 2015. ACM.

[43] L. Tseng and N. H. Vaidya. Iterative approximate Byzantine consensus under a generalized fault model. In *In International Conference on Distributed Computing and Networking (ICDCN)*, January 2013.

[44] N. H. Vaidya. Matrix representation of iterative approximate Byzantine consensus in directed graphs. *CoRR*, Mar. 2012.

[45] N. H. Vaidya. Iterative Byzantine vector consensus in incomplete graphs. In *In International Conference on Distributed Computing and Networking (ICDCN)*, January 2014.

[46] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. In *Proceedings of the thirty-first annual ACM symposium on Principles of distributed computing*, PODC '12. ACM, 2012.

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

[48] J. Wolfowitz. Products of indecomposable, aperiodic, stochastic matrices. In *Proceedings of the American Mathematical Society*, volume 14, pages 733–737, 1963.

[49] H. Zhang and S. Sundaram. Robustness of complex networks with implications for consensus and contagion. In *Proceedings of CDC 2012, the 51st IEEE Conference on Decision and Control*, 2012.